VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Spektrální trasování

Spectral ray tracing

2014 Bc. Jan Šurík

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Jan Šurík**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Spektrální trasování
Spectral Ray Tracing

Zásady pro vypracování:

Realističtějšího vzhledu u syntetizovaných obrazů lze dosáhnout také pomocí spektrálního trasování, při kterém je brána v úvahu vlnová povaha světla. To, mimo jiné, přináší potřebu charakterizovat optické vlastnosti materiálů v závislosti na vlnové délce interagujícího záření. Jelikož potřeba výrazně vyššího počtu vzorků prodlužuje dobu výpočtu, je vhodné uvažovat s implementaci trasování pomocí CUDA nebo OptiX. Implementaci proveďte v jazyce C++ za dodržení Google C++ Style Guide a kód důkladně zdokumentujte pomocí nástroje Doxygen.

1. Seznamte se s principy spektrálního trasování.
2. Navrhněte struktury pro reprezentaci paprsků a materiálů v závislosti na vlnové délce.
3. Implementujte spektrální trasování pomocí C++/CUDA.
4. Funkčnost demonstrujte na několika vhodných scénách.

Seznam doporučené odborné literatury:

[1] Morley, R. Keith, Shirley, Peter. Realistic Ray Tracing. 2nd edition. 2003. 235 s.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Tomáš Fabián**

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2014 ..... Jan Šmíd .....

Rád bych poděkoval panu Ing. Tomáši Fabiánovi za odborné vedení a bystré podněty při zpracovávání tématu. Dále pak všem za veškerou podporu, kterou mi poskytli.

Abstrakt

Práce se zabývá metodou spektrálního sledování paprsků a jejím přínosem při syntetizování obrazů. Jsou představeny optické jevy a vlastnosti reálných materiálů závislé na vlnové délce světla. Praktická část se věnuje implementací jak pro CPU, tak i pro GPU s využitím NVIDIA OptiX SDK.

Klíčová slova: spektrální trasování, sledování paprsku, fyzikálně založené renderování, nvidia optix, diplomová práce

Abstract

Thesis consults spectral ray tracing and its benefits toward synthetic image generation. Optical laws and wavelength dependant properties of materials are introduced. Practical part pursues implementation for both CPU and GPU, later using NVIDIA OptiX SDK.

Keywords: spectral ray tracing, ray tracing, physically based rendering, nvidia optix, master thesis

Seznam použitých zkratek a symbolů

CPU	–	Central Processing Unit
GPU	–	Graphics Processing Unit
CUDA	–	Compute Unified Device Architecture
SDK	–	Software Development Kit
GLSL	–	OpenGL Shading Language
OMP	–	Open Multi-Processing
MPI	–	Open Message Passing Interface

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Počítačová syntetizace fotorealistických obrazů je oblastí počítačové grafiky, která dovoluje vykreslit jakoukoliv uměle vytvořenou scénu tak, jak by vypadala v reálném světě. Ovšem i přes skvělé výsledky je stále potřeba tuto oblast zdokonalovat, jelikož se kvůli výpočetním nárokům stále zjednodušují a aproximují patřičné fyzikální zákony. V tomto textu se podíváme na metodu zvanou spektrální ray tracing (chcete-li spektrální trasování), která umožňuje vypočítat jevy vznikající na rozhraní různých optických prostředí.

Nejprve si představíme metodu běžného ray tracingu, z něj odvozeného spektrálního trasování a jaké toto vylepšení přináší výhody a nevýhody. Dále se podíváme na teoretické podklady, které je pro implementaci tohoto algoritmu potřeba znát. Zde si popíšeme co je vlastně světlo, jaké optické vlastnosti mají materiály v reálném světě a jaké optické zákony jich využívají.

V praktické části je popsáno NVIDIA OptiX SDK, které nabízí využití grafických akcelerátorů s podporou technologie CUDA pro výpočet algoritmů ray tracingu. Také se budeme zabývat implementací spektrálního trasování, a to jak na CPU, tak i na GPU s využitím výše zmíněného NVIDIA OptiX SDK. Nakonec tyto implementace podrobíme testování výkonu.

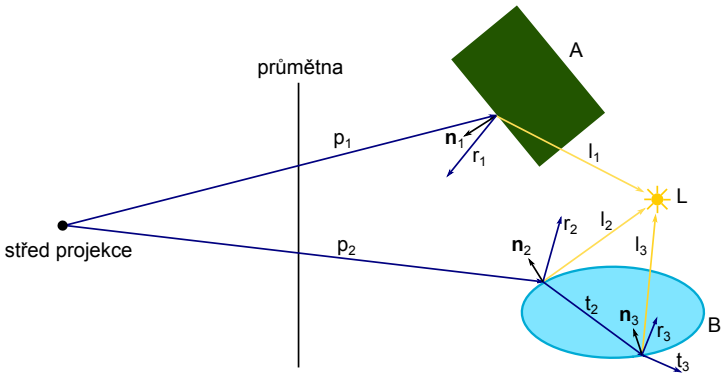
2 Ray tracing

Metoda sledování paprsků, neboli ray tracing, je velmi populárním přístupem při syntetizaci počítačových obrazů. Je to hlavně díky tomu, že v umělých scénách umožňuje vypočítat šíření světla s ohledem na veškeré optické zákony. Proto jsou tyto syntetizované obrazy velmi kvalitní. Výpočet probíhá sledováním šíření paprsků světla ve scéně, tedy paprsků, které vycházejí ze světelného zdroje. Ve scéně je pak umístěna kamera s průmětnou. Kamera představuje oko pozorovatele a průmětna výsledný obraz, do kterého se ukládají barvy z paprsků, které do ní dopadnou. Při tomto přístupu se však počítá spousta paprsků, které do průmětny nikdy nedopadnou a pouze prodlužují výpočet bez jakéhokoliv přispění do obrazu. Z tohoto důvodu se trasování řeší jako zpětné sledování paprsků, kdy jsou paprsky vysílány z kamery skrz průmětnu do scény a počítá se pouze jejich šíření scénou. Tak se zajistí, že žádný paprsek není sledován zbytečně.

Algoritmus se zpravidla implementuje rekurzivně, tj. rekurzivní ray tracing. Je to proto, že bez rekurze by výsledky trasování scény byly značně neúplné, chyběly by odrazy a například sklo by nebylo průhledné. Proto se při dopadu každého paprsku vyšle z místa dopadu další paprsek, jehož směr se určí podle geometrických a optických zákonů. Tak nám vznikne rekurze, která zaručí kvalitnější výsledek. Zároveň se však trasování stává časově velmi náročným a proto nevhodným pro simulace v reálném čase, ačkoliv současné technologie již umožňují i tento přístup.

2.1 Whittedův ray tracing

Whittedův ray tracing je asi nejznámější algoritmus pro rekurzivní sledování paprsků. Z kamery se skrz průmětnu vysílají tzv. primární paprsky. Pro ně zjišťujeme průsečíky s objekty scény. Z každého tohoto průsečíku se poté vysílají sekundární paprsky. Těmito paprsky jsou shadow ray - paprsek vyslaný ke zdroji světla, ověřuje, zda je průsečík daným světlem osvícen. Dále pak reflected ray - paprsek odražený od povrchu dále do scény a transmitted ray - paprsek, který vlivem lomu světla na rozhraní dielektrických prostředí prochází tělesem. Rekurzivně se pak tento postup opakuje pro každý odražený a lomený paprsek. Výsledek tohoto vysílání paprsků si lze představit jako strom.



Obrázek 1: Ukázka průchodu Whittedova algoritmu scénou

Na obrázku je tento postup znázorněn. Paprsky p jsou vysílány ze středu projekce průmětnou do scény. Ty pak narazí na objekty A - neprůhledné těleso nebo B - dielektrické těleso. Od nich se nadále odráží (r) nebo lámou (t) podle normály n v místě dopadu. V každém místě dopadu je pak kontrolováno přímé osvětlení l.

Důležitým krokem je určit maximální hloubku rekurze. Určitě nechceme, aby se algoritmus zanořoval do nekonečna. Proto se nejčastěji předem definuje požadovaná maximální hloubka rekurze, resp. jiná ukončovací kritéria. Následující pseudokódy byly převzaty z textu [1] a upraveny do jazyka C++

```
{Ray je struktura obsaujici pocatek a smer paprsku.}
for ( int i = 0; i < image_width; i++ )
{
  for ( int j = 0; j < image_height; j++ )
  {
    Ray ray = paprsek z kamery do pixelu (i, j)
    Image(i, j) = RayTrace( ray, 0 );
  }
}
```

Výpis 1: Pseudokód pro Whittedův ray tracing

z kódu vyplývá, že většinu práce obstarává funkce `RayTrace`. Ta pak vypadá následovně

```
{Ray je struktura obsaujici pocatek a smer paprsku.}
{Color je vektor barevnych slozek r, g, b.}
Color RayTrace( Ray ray, int depth )
{
  Jestlize depth > max_depth vrat (0, 0, 0);
  Najdi nejblizsi prusecik paprsku s objekty sceny;
  Jestlize prusecik neexistuje, vrat (0, 0, 0);
  X = prusecik nejblizsi pocatku paprsku;
  Ray reflected = paprsek odrazeny v X;
  Color Ir = RayTrace( reflected, depth + 1 );
  Ray refracted = paprsek lomeny v X;
  Color It = RayTrace( refracted, depth + 1 );
  Color Il = intenzita v bode X od svetelnych zdroju;
  Color c = Il + KrIr + KtIt;
  vrat c;
}
```

Výpis 2: Pseudokód funkce RayTrace

2.2 Spektrální ray tracing

Výsledky běžného ray tracingu, ačkoliv jsou dobré, nejsou stále opticky zcela správné. Je to dáno mimo jiné i tím, že je ignorována vlnová povaha světla a vlastností materiálů. Tento nedostatek se snaží odstranit metody spektrálního trasování. Zde se právě vlnová povaha světla uvažuje a díky tomu tento přístup dokáže simulovat i vlnově závislé optické jevy způsobené chromatickou disperzí. Takže lze trasovat například duhu nebo chromatickou aberaci. Nejjednodušším způsobem, jak spektrální trasování implementovat, je úprava Whittedova algoritmu.

3 Teoretické podklady

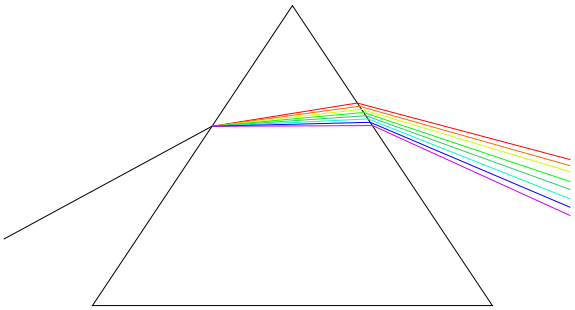
Abychom dokázali spektrální ray tracing implementovat, je potřeba seznámit se s povahou světla a jeho chováním v reálném světě. Jelikož je však světlo z pohledu fyziky značně komplikované, budeme se zabývat pouze vlastnostmi, které jsou relevantní pro spektrální trasování. Dále si představíme akcelerační struktury a barevné prostory, které také tvoří důležitou část implementace.

3.1 Světlo

Světlo jako takové je elektromagnetické záření, bez ohledu na jeho vlnovou délku. Nás však zajímá viditelné světlo, tedy elektromagnetické záření o vlnové délce v rozmezí přibližně 380nm – 780nm. Tyto vlnové délky je lidské oko schopno vnímat a zpracovávat je v obraz okolního světa. Z historického pohledu se chápání světla značně měnilo. Například Newton věřil, že světlo je mechanický proud částic. Tyto částice byly popsány jako tzv. černá tělesa, tedy tělesa, která žádné světlo neodráží. Tato teorie však nedokázala uspokojivě objasnit lom světla. Později přišla vlnová teorie, která přinesla nové poznatky. Mezi ně patří například barva světla nebo jeho polarizace. Slabinou této teorie je však předpoklad, že vlny k přenosu potřebují nějaké médium. Proto se světlo nejčastěji uvažuje jako obojí, částice i vlna. Nyní se blíže podíváme na šíření světla prostředím. Dochází zde k důležitým jevům.

3.1.1 Lom světla

Jedním z těchto jevů je lom světla. K tomu dochází v případě, že vlna přechází z jednoho optického prostředí do jiného. Typickým příkladem je například rozhraní vzduch - sklo. Při lomu světla zároveň dochází k jeho zpomalení oproti rychlosti ve vakuu. Toto zpomalení je vyjádřeno indexem lomu (kapitola 3.2.5). Tím také dochází ke změně vlnové délky. Díky tomu lze například bílé světlo pomocí hranolu rozložit na jeho barevné složky. Tomuto jevu se také říká disperze světla (chromatická disperze) a je tedy závislá na vlnové délce. Dielektrické materiály, jak si řekneme později, mají pro různé vlnové délky různý index lomu, což je příčina zmíněné disperze. Přímým důsledkem lomu světla je například známé zkreslení polohy objektů umístěných ve vodě.



Obrázek 2: Ukázka chromatické disperze

Vlivem chromatické disperze pak může v závislosti na vlastnostech disperzního prostředí docházet k dalším jevům, které se mohou projevovat jak uvnitř, tak vně tohoto prostředí. První z těchto jevů je *chromatická aberace* a je to jev tzv. vnitřní (vidíme jej uvnitř prostředí). Tento jev se vyskytuje u čočky, popř. složitějších optických soustav čoček. Dochází zde k tomu, že ohnisková vzdálenost čočky se mění v závislosti na vlnové délce. To je způsobeno tím, že index lomu zpravidla není pro všechny vlnové délky konstantní. Výsledkem tohoto jevu je barevné lemování v místech, kde sousedí světlé a tmavé části obrazu. Druhý jev se projevuje navenek a známe jej například jako duhu při dešti. Opět při vstupu světla do prostředí (v tomto případě vody) dochází k jeho rozptylu. Tyto rozptýlené paprsky světla se pak uvnitř kapky opět odrazí ven, kde je pak můžeme spatřit v atmosféře jako duhu. O této duze se říká, že vznikla odrazem. Je to proto, že vidíme rozptýlené světlo odražené uvnitř kapek, kdy slunce musíme mít za sebou. Dalším případem je duha vzniklá při průchodu světla hranolem. Princip je stejný jako u atmosférické duhy, pouze s tím rozdílem, že duhu nevidíme díky odrazu uvnitř hranolu, ale jsou to paprsky, které hranolem projdou. Při vstupu do prostředí je světlo rozptýleno a při výstupu je ještě jednou lomeno. K rozptylu už však nedochází, protože paprsky již většinou přenáší jednu vlnovou délku. Tato duha tedy vzniká refrakcí světla.

3.1.2 Absorpce světla

Dalším jevem, ke kterému dochází při šíření světla, je absorpce. Při dopadu světla na povrch dochází k jeho částečnému pohlcení, přičemž se povrch zároveň slabě zahřeje. Proto se světlo odražené od povrchu již nejeví tak jasné, jako jeho původní zdroj. Každý atom, ze kterého se povrch skládá, přitom pohlcuje určitou vlnovou délku. Barva povrchu tedy záleží na jeho složení, tj. na tom, které vlnové délky pohlcuje a které nikoli. Absorpce se obecně dělí na 3 skupiny

1. **Selektivní absorpce** - dochází k pohlcování pouze určité části spektra. Ve spektru světla se po absorpci nenachází frekvence, které byly absorbovány povrchem. Selektivní absorpce se vyskytuje u většiny látek a tím zapříčiňuje jejich obarvení.

2. **Neutrální absorpce** - světlo je v daném rozsahu pohlcování stejnou mírou.

3. **Spojitá a čárová absorpce** - spojitá absorpce znamená, že je světlo pohlcováno ve všech vlnových délkách. Naopak čárová absorpce popisuje pohlcování světla pouze v určitých spektrálních čarách (způsobuje tak nedostatek fotonů v úzkém frekvenčním pásmu).

Absorpcí světla při průchodu dielektrickým prostředím se dále zabývá kapitola 3.6 popisující Beer-Lambertův zákon.

3.1.3 Veličiny

Nyní se podívejme na vlastnosti světla, které nás zajímají. Jsou jimi amplituda, vlnová délka a polarizace.

Amplituda - zastupuje energii přenášenou na dané vlnové délce, tedy energii, kterou pak lidské oko vnímá jako jas barvy.

Vlnová délka - každá vlnová délka viditelného světla je lidským okem vnímána jako určitá barva. Různé barevné odstíny pak vznikají díky zastoupení více vlnových délek ve světle dopadajícím na sítnici oka.

Polarizace - polarizací obecného elektromagnetického vlnění rozumíme, že výchylky vlny probíhají pouze v určitém směru vůči směru jejího šíření. Světlo je však obecně nepolarizované, tedy kmitá chaoticky v různých směrech. K polarizaci světla může dojít například při dopadu vlny na rozhraní dvou dielektrických prostředí nebo použitím polarizačního filtru. Uvažujme rovinu, která je definována směrem šíření světla a vektoru, jenž je kolmý na povrch dopadu, tedy normálu daného bodu. Pak může dojít k polarizaci světla, resp. složek jeho elektromagnetického pole. Prvním případem je světlo, které je polarizováno paralelně s rovinou dopadu. To označujeme symbolem p - $parallel$. Druhým výsledkem je světlo polarizované kolmo na danou rovinu. To je označeno s - z německého $senkrecht$, což v překladu znamená "kolmý". V praktické implementaci jsou však tyto případy polarizace uvažovány pouze během výpočtu odrazivosti a transmisivity povrchu ve Fresnelových rovnicích, viz kapitola 3.5.

3.2 Fotometrie

Fotometrie je věda zabývající se měřením světla pomocí různých veličin a jak je vnímá lidské oko. To není stejně citlivé na všechny vlnové délky. Fotometrie se toto snaží zohlednit vážením energie na každé vlnové délce hodnotou, která představuje citlivost oka pro danou vlnovou délku.

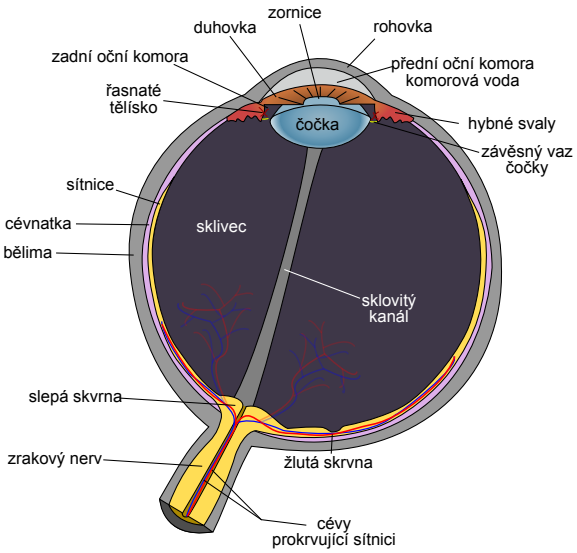
Světově uznávanou organizací, která se fotometrií zabývá, je komise CIE - Commission internationale de l'éclairage, v překladu Mezinárodní komise pro osvětlování. Do jejího zaměření patří zkoumání světla, osvětlování, barev a barevných prostorů. V současnosti mají 7 aktivních divizí, jejichž bližší popis je k dispozici online [2]. Mezi jedny z nejdůležitějších přínosů této komise patří definování barevných prostorů CIE XYZ a CIE RGB, referenčních světelných zdrojů, standardního pozorovatele a funkcí poskytujících matematický popis chromatické odezvy tohoto pozorovatele. Komise CIE dále poskytuje různá kolorimetrická data volně ke stažení [3].

Další oblastí, která se zabývá barvami, je kolorimetrie. Ta se snaží kvantifikovat a fyzikálně popsat lidské vnímání barev, nejčastěji vztažené k CIE 1931 XYZ barevnému prostoru. K těmto měřením slouží různé kolorimetrické nástorje.

Asi nejvýznamnější stroj je tristimulový kolorimetr. Ten se nejčastěji používá pro kalibraci zobrazovacích zařízení. Toho dosahuje pomocí vzorkování viditelného spektra přes fotodetektory, jako jsou například fotorezistory.

3.2.1 Lidské oko

Lidské oko je stavěno tak, aby co nejlépe zaostřovalo paprsky světla na sítnici. Proto jsou všechny jeho části, kterými světlo prochází, průhledné, aby nedocházelo k nežádoucímu rozptylu paprsku.



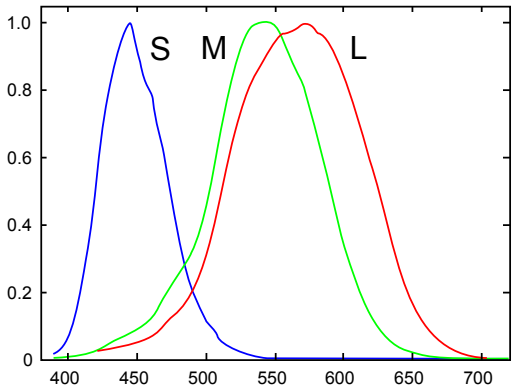
Obrázek 3: Schéma lidského oka, zdroj: *Wikipedia* [9]

Světlo do oka vstoupí přes rohovku a skrz zornici dopadá na čočku. Rohovka a čočka mají za úkol paprsek spojit a zaostřit na sítnici. Zornice se pomocí duhovky stahuje či roztahuje, čímž omezuje množství světla, které do oka dopadne. Také čočka pomocí svalů mění tvar, aby co nejlépe zaostřila paprsky tak, aby se sbíhaly na sítnici. Sítnice pak obsahuje čípky a tyčinky, které jsou světlem stimulovány a tyto stimuly pak zrakovým nervem putují do mozku. Tyčinky dokáží rozlišovat pouze odstíny šedi a je jich kolem 130 milionů. Díky jejich vyšší citlivosti na světlo umožňují vidět v horších světelných podmínkách. Proto v noci není člověk schopen příliš rozlišovat barvy. Zajímavější jsou však čípky. Ty jsou rozděleny do 3 druhů (v závislosti na vazbách vitaminu A), každý pak má jinou citlivost na červené, zelené a modré světlo. Nejvíce jsou nakupeny v tzv. žluté skvrně, kde se zároveň nenachází žádné tyčinky. Oko také obsahuje slepé místo. Je to tam, kde v sítnici vystupuje zrakový nerv. Díky jeho umístění v oku a vzájemné pozici obou očí jsou však tyto slepá místa překryta oblastí vnímání druhého oka a vidění tedy nemá žádný slepý bod.

3.2.2 Tristimulus

V předchozí kapitole jsme se zmínili o čípcích a jejich různém vnímání spektra. Každý druh těchto buněk má v určité části spektra nejvyšší citlivost. V krátkých vlnových délkách (S) v rozmezí 420 – 440 nm, střední vlnové délky (M) 530 – 540 nm a dlouhé (L) 560 – 580 nm.

Díky tomu lze jakýkoliv barevný vjem docílit pomocí tří hodnot S, M a L, které představují stimulaci jednotlivých druhů čípků. Těmto hodnotám se říká tristimulus a můžeme

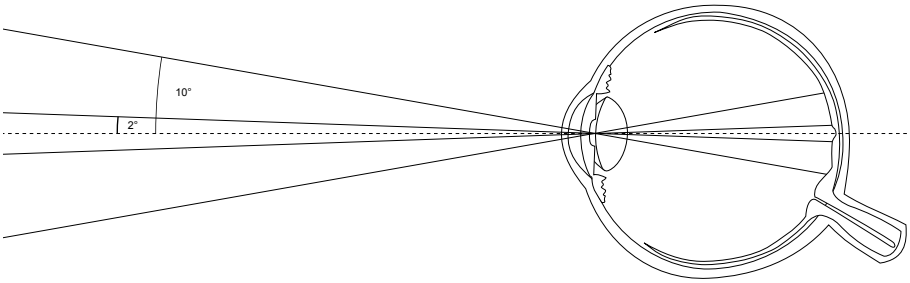


Obrázek 4: Spektrální citlivost čípků, zdroj: *Wikipedia* [10]

je zobrazit ve třírozměrném prostoru, zvaném LMS barevný prostor. Tento tristimulus ve spojení s barevným prostorem můžeme chápat jako zastoupení tři primárních barev v aditivním barevném modelu.

3.2.3 Standardní pozorovatel

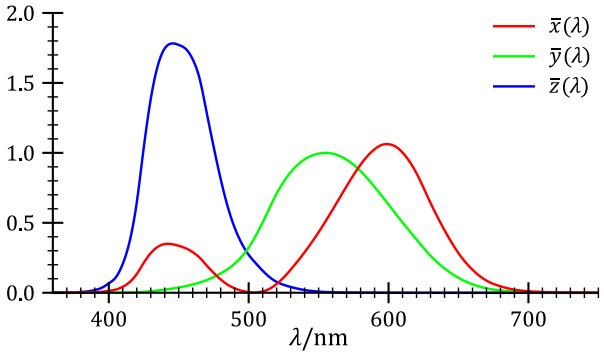
Standardní pozorovatel je jeden z asi nejdůležitějších produktů fotometrie komise CIE. Je to soubor funkcí, který popisuje, jak čípky v oku průměrného člověka vnímají různé vlnové délky viditelného spektra. Přesněji se jedná o soubor průběhů 3 funkcí, které popisují vnímání světla pro každý ze 3 druhů čípků, tedy pro červenou, zelenou a modrou barvu.



Obrázek 5: Zorné úhly standardního pozorovatele

Průběh těchto funkcí je závislý na rozšíření zornic, tedy kde na sítnici světlo dopadá. Roku 1931 CIE definovala standardního pozorovatele [4] pro 2° zorný úhel, kde světlo dopadá výhradně do žluté skvrny. Tato specifikace se také nazývá *CIE* 2° *Standard Observer*. Později, v roce 1964, bylo definováno rozšíření funkce pozorovatele pro 10° zorný úhel, takzvaný doplňkový pozorovatel, kdy světlo dopadá i mimo žlutou skvrnu. Motivací této definice bylo vědomí, že světlo v oku běžně dopadá na celou sítnici.

Na obrázku 6 je znázorněn průběh funkcí standardního pozorovatele pro 2° zorný úhel. Funkční hodnoty představují citlivost čípků na různé vlnové délky. Barevné roz-



Obrázek 6: Funkce standardního pozorovatele s 2° zorným úhlem, zdroj: *Wikipedia* [10]

lišení pak představuje tři druhy čípků podle barev, které vnímají. Funkční hodnoty jsou diskretizovány v rozsahu 380 – 780 nm každých 5 nm, čímž přes viditelné spektrum dávají 81 vzorků. Tyto funkce se poté používají k převodu spektrálních vzorků do barevného prostoru CIE XYZ, o čemž se více zmíníme v kapitole 3.8.

3.2.4 Standardní zdroj světla

Standardní zdroj světla je teoretický zdroj světla, jehož spektrální profil byl zveřejněn. Zpravidla se používá jako referenční bod při porovnávání obrazů. Prvními třemi byly zdroje A, B a C zveřejněné roku 1931 [4]. Ty měly představovat, ve stejném pořadí, běžnou žárovku, přímé sluneční světlo v poledne a průměrné denní světlo. Zdroje B a C byly odvozeny z A použitím kapalinových filtrů. Jsou však špatnou aproximací reálných zdrojů světla, a proto ustoupily sérii D.

Série D je série zdrojů navržená tak, aby odpovídala přirozenému dennímu světlu v různých částech světa. Jednotliví zástupci mají určenou barevnou teplotu, což je teplota, kterou by měly mít fotony z pohledu částic - černé těleso. Jedná se tedy o charakteristiku spektrální distribuce bílého světla. Tato teplota je označena dolním indexem. Mezi známé zástupce patří D_{50}, D_{55}, D_{65} a D_{75}. Nás nejvíce zajímá D_{65}. Tento zdroj popisuje polední slunce v západní a střední Evropě. Barevná teplota byla určena na 6504°K.

Dalším standardním zdrojem je E, který má na všech vlnových délkách stejnou energii. Tento zdroj se nepovažuje za zdroj černých těles, nemá proto specifikovanou teplotu. Obecně se ale dá aproximovat pomocí D_{55}.

Posledním zástupcem je série F. Tato série zastupuje různé zářivky. F1 až F6 jsou běžné zářivky. F7 až F9 jsou širokopásmové zářivky, které vyzařují v celém spektru a F10 až F12 jsou úzkopásmové, svítící převážně v oblastech červené, modré a zelené barvy.

3.2.5 Optické vlastnosti materiálů

Chceme-li syntetizovat obraz, potřebujeme znát optické vlastnosti materiálů ve scéně, abychom je dokázali správně zobrazit. Požadované vlastnosti se však mezi běžným a

spektrálním trasováním liší, proto si obě varianty představíme.

Při běžném trasování uvažujeme pouze 3 vlnové délky, resp. barvy, které lidské oko dokáže vnímat. Jsou to červená, zelená a modrá, primární barvy RGB prostoru. Proto i vlastnosti materiálů jsou omezeny na tyto 3 barevné složky.

- Ambientní koeficient - k_a - 3 hodnoty pro ambientní odrazivost
- Difuzní koeficient - k_d - 3 hodnoty pro difuzní odrazivost
- Spekulární koeficient - k_s - 3 hodnoty pro spekulární odrazivost
- Svítivost - s - konstanta určující ostrost odlesku na povrchu tělesa
- Vyzařování - 3 složky barvy světla, které objekt vyzařuje v případě, že je zdrojem světla
- Neprůhlednost - konstanta v rozsahu $\langle 0, 1 \rangle$, určuje, nakolik je materiál průhledný
- Index lomu - konstanta určující index lomu materiálu

Tyto konstanty pak používáme pro výpočet Phongova stínování, resp. u sledování tras (path tracing) v distribučních funkcích. Všechny tyto koeficienty jsou však pouze aproximací skutečnosti. V reálném světě se většina těchto hodnot nevyskytuje.

Nyní se podívejme na vlastnosti požadované při spektrálním trasování. Ty už jsou fyzikálně mnohem přesnější, protože se neomezují na tři vlnové délky. Zpravidla bývají vzorkovány tak, aby odpovídaly diskrétním vzorkům standardního pozorovatele. Z hlediska optiky potřebujeme znát pouze spektrální charakteristiku pro barvu, vyzařovanou barvu a index lomu. Průhlednost materiálu se určí pomocí Fresnelových rovnic a Beer-Lambertova zákona. Jak bylo zmíněno výše, zbarvení objektů způsobuje *selektivní absorpce*, potřebujeme tedy znát její průběh přes námi zvolenou část spektra. Obdobně to platí pro vyzařovanou barvu v případě, že materiál může být také zdrojem světla. V tom případě potřebujeme znát energie vyzařované na jednotlivých vlnových délkách. Nyní se blíže podívejme na index lomu n:

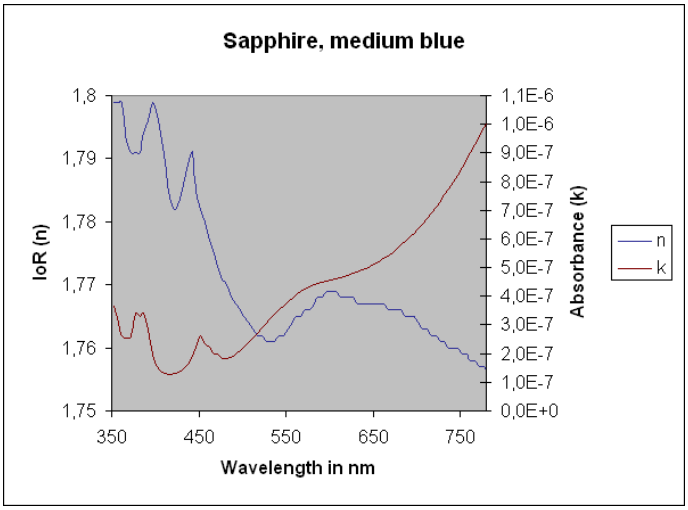
$$n = \frac{c}{v} \,, \tag{1}$$

kde $c \approx 300000$ kms^{-1} je rychlost světla ve vakuu a v je rychlost šíření v prostředí. Z toho vyplývá, že se jedná o poměr mezi rychlostmi světla ve vakuu a měřeném prostředí, tedy kolikrát pomaleji se světlo v daném prostředí pohybuje. Nicméně, vzhledem k faktu, že při šíření světla dochází zároveň k jeho absorpci, je vhodné definovat tzv. *komplexní index lomu*:

$$\tilde{n} = n + ik \,. \tag{2}$$

Proměnná n je zde index lomu a k je absorbance prostředí. Díky této definici jsme schopni jednodušeji popsat vlastnosti dielektrických prostředí. Tyto vlastnosti jsou z pohledu spektrálního trasování dostačující. K výpočtu osvětlení pak slouží distribuční funkce.

Na obrázku 7 je ukázka závislosti indexu lomu a absorbance safíru na vlnové délce světla. Jak je patrné, změny mohou být vcelku razantní. Tyto změny pak jsou příčinou různých obrazců, které světlo procházející tímto prostředím vytváří.



Obrázek 7: Index lomu a absorbance safíru, zdroj: *BayleeOnline* [12]

3.3 Distribuční funkce

Distribuční funkce v počítačové grafice představují matematický popis, jak se světlo odráží od povrchu objektů. Přesněji jde o vyjádření poměru světla, které se v daném bodě odráží v určitém směru vůči světlu, které na daný bod dopadá z okolí:

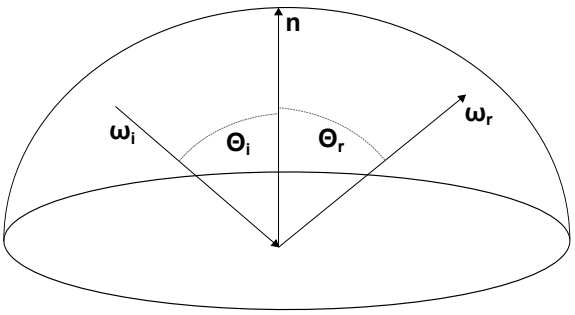
$$f\left(\boldsymbol{\omega_i}, \boldsymbol{\omega_r}, \lambda\right) = \frac{dL_r\left(\boldsymbol{\omega_r}, \lambda\right)}{dE\left(\boldsymbol{\omega_i}, \lambda\right)} = \frac{dL_r\left(\boldsymbol{\omega_r}, \lambda\right)}{L_i\left(\boldsymbol{\omega_i}, \lambda\right) \cos\theta_i d\boldsymbol{\omega_i}}\ , \tag{3}$$

kde E je intenzita záření a L je záře na vlnové délce λ, θ_i je úhel mezi dopadajícím paprskem a normálou v bodě x, $\boldsymbol{\omega}$ představuje paprsek. Index i označuje příchozí směr a r odražený směr.

Abychom však mohli tuto distribuční funkci použít při výpočtu osvětlení, musíme znát její hodnoty pro všechny dvojice dopadajících a odražených paprsků. K tomu slouží přístroj zvaný *gonioreflektometr*. Ten se skládá ze světelného zdroje pro osvícení materiálu, jehož distribuční funkce se měří, a senzoru, který zachytává světlo odražené materiálem. Zdroj světla i senzor jsou na pohyblivých ramenech, díky nimž lze měřit veškeré kombinace příchozích a odražených směrů paprsků pod různými úhly (proto je v názvu gonio). Takhle vzniklá sada dat by však byla příliš velká. Proto se z nich poté analyticky získává matematický předpis distribuční funkce měřeného materiálu. Celý proces je však vcelku zdlouhavý.

Distribuční funkce se dělí do tří skupin:

- BRDF - Bidirectional Reflectance Distribution Function - modeluje odraz světla od povrchu. Typickým zástupcem je Lambertova reflektance.
- BTDF - Bidirectional Transmission Distribution Function - modeluje průchod světla povrchem u dielektrických materiálů, jako je například sklo.



Obrázek 8: Hemisféra kolem normály v bodě dopadu

- BSDF - Bidirectional Scattering Distribution Function - je zobecněním BRDF a BTDF, které spojuje do jedné. Slouží tedy jako obecný popis jakéhokoliv povrchu, bez ohledu na to, zda světlo pouze odráží nebo jej i propouští. V praxi se však pro výpočty používají BRDF a BTDF.

Aby byla distribuční funkce fyzikálně korektní, musí splňovat tyto podmínky:

1. pozitivita - funkční hodnota je pro jakoukoliv dvojici směrů větší nebo rovna nule.
2. Helmholtzova reciprocita - $f(\boldsymbol{\omega_i}, \boldsymbol{\omega_r}, \lambda) = f(\boldsymbol{\omega_r}, \boldsymbol{\omega_i}, \lambda)$
3. zákon zachování energie

Konkrétních modelů distribučních funkcí existuje celá řada. Rozdíly mezi nimi jsou převážně v tom, v jaké míře jsou fyzikálně korektní. Nyní si představíme ty, které byly v práci implementovány.

Difuzní BRDF - modeluje matný povrch, jako je například běžná školní tabule. Světlo se na tomto povrchu odráží všemi směry, jeho intenzita je určena Lambertovým kosinovým zákonem:

$$I_d = \mathbf{n} \cdot \mathbf{l} C I_i \,, \tag{4}$$

kde I_d je intenzita odraženého světla, C je barva povrchu, I_i je intenzita dopadajícího světla a $\mathbf{n} \cdot \mathbf{l}$ vyjadřuje kosinus úhlu mezi normálou a směrem dopadajícího světla. Z tohoto vztahu pak vyplývá distribuční funkce:

$$f(\boldsymbol{\omega_i}, \lambda) = C(\lambda)\, \mathbf{n} \cdot \boldsymbol{\omega_i} \,. \tag{5}$$

Zrcadlová BRDF - případ ideálně lesklého povrchu. Světlo je zde odraženo pouze v jediném směru podle zákona odrazu, aniž by docházelo k jeho pohlcení. Vztah pro výpočet je tedy jednoduchý:

$$I_d = I_i \,. \tag{6}$$

V tomto případě tedy nelze hovořit o distribuční funkci v pravém slova smyslu, protože dopadající světlo se kompletně odrazí v jediném směru a není nijak distribuováno do okolí.

Phongova BRDF - představuje Phongův osvětlovací model, což je empirický model lokálního osvětlení:

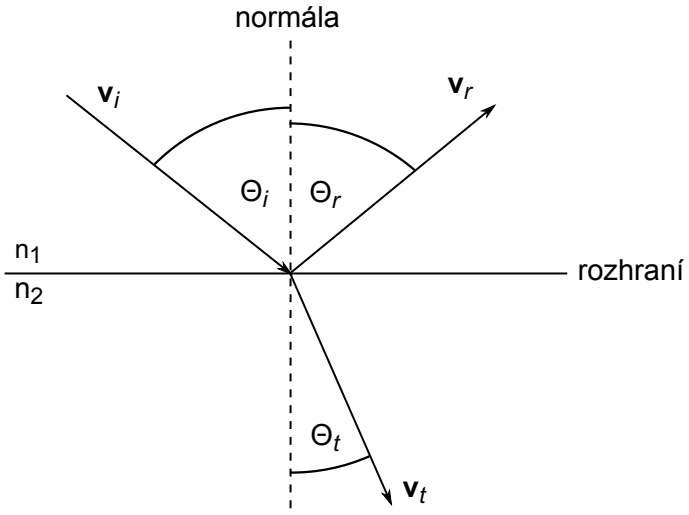
$$f\left(\boldsymbol{\omega_i}, \boldsymbol{\omega_r}, \lambda\right) = C\left(\lambda\right) + k_s \frac{\left(R\left(\boldsymbol{\omega_r}, \mathbf{n}\right) \cdot \boldsymbol{\omega_i}\right)^e}{\mathbf{n} \cdot \boldsymbol{\omega_i}} \ , \tag{7}$$

kde k_s je spekulární koeficient, e určuje ostrost spekulárního odrazu a $R\left(\boldsymbol{\omega_r}, \mathbf{n}\right)$ vypočte odraz paprsku $\boldsymbol{\omega_r}$ podle normály $\mathbf{n}$. Spekulární koeficient a ostrost spekulárního odrazu jsou v reálném světě určeny hrubostí povrchu. V implementaci byly nahrazeny vhodně zvolenými konstantami.

Dielektrická BSDF - modeluje chování světla při dopadu na rozhraní dvou dielektrických prostředí. Vzhledem k tomu, že zde může docházet k odrazu i lomu paprsku, je tato distribuce popsána jako BSDF. Reflexi a transmisi pak řeší Snellův zákon a Fresnelovy rovnice.

3.4 Snellův zákon

Snellův zákon popisuje chování paprsku na rozhraní dvou optických prostředí, například vzduch - sklo. Přesněji udává vztah mezi směrem dopadajícího paprsku a paprsků odražených od povrchu a lomených do tělesa v případě dielektrických materiálů.



Obrázek 9: Rozhraní optických prostředí

Na obrázku 9 je znázorněna situace, kdy paprsek dopadne na rozhraní dvou optických prostředí. V závislosti na materiálu zde dochází jak k odrazu, tak lomu paprsku. Dle zákona odrazu platí, že $\theta_i = \theta_r$. Na tomto rozhraní může dojít ke dvěma jevům. Jsou jimi totální interní odraz a totální transmise.

Totální interní odraz nastává v případě, že úhel dopadu paprsku θ_i je větší než tzv. kritický úhel

$$\theta_{C,\lambda} = \sin^{-1}\left(\frac{\theta_{t,\lambda}}{\theta_i}\right) \ . \tag{8}$$

V tomto případě nedochází k lomu světelného paprsku a ten se od povrchu pouze odrazí.

Totální transmise je jev, kdy paprsek na rozhraní dopadá pod tzv. Brewsterovým úhlem

$$\theta_{B,\lambda} = \tan^{-1}\left(\frac{\theta_t}{\theta_i}\right) . \tag{9}$$

V tomto případě dochází k úplnému průchodu paprsku rozhraním a není tedy žádný odraz.

Tyto jevy je potřeba si při implementaci pohlídat, jelikož mají také vliv na reálnost výsledku. Nyní můžeme určit směr odraženého a lomeného paprsku. Nechť $\mathbf{v}_i$ je směrový vektor dopadajícího paprsku a $\mathbf{n}$ je normálový vektor v bodě dopadu, pak směrový vektor odraženého paprsku vypočteme vztahem

$$\mathbf{v}_r = \mathbf{v}_i + 2\cos\left(\theta_i\right)\mathbf{n} \tag{10}$$

a směrový vektor lomeného paprsku, v závislosti na vlnové délce

$$\mathbf{v}_{t,\lambda} = \left(\frac{n_{1,\lambda}}{n_{2,\lambda}}\right)\mathbf{v}_i + \left(\frac{n_{1,\lambda}}{n_{2,\lambda}}\cos\theta_i - \cos\theta_{t,\lambda}\right)\mathbf{n} . \tag{11}$$

Pro výpočet vztahu (11) potřebujeme znát ještě úhel lomeného paprsku, resp. jeho cosinus

$$\cos\theta_{t,\lambda} = \sqrt{1 - \left(\frac{n_{1,\lambda}}{n_{2,\lambda}}\right)^2\left(1 - \left(\cos\theta_i\right)^2\right)} . \tag{12}$$

3.5 Fresnelovy rovnice

Fresnelovy rovnice popisují chování světla na rozhraní dvou optických prostředí z pohledu dělení jeho energie. S jejich pomocí tedy spočteme transmisivitu a reflektivitu dielektrického povrchu. Jak už víme, světlo může být polarizováno dvěma způsoby. Pro každý z nich je tedy potřeba spočítat reflektivitu podle následujících vztahů

$$R_{S,\lambda} = \left|\frac{n_{1,\lambda}\cos\left(\theta_i\right) - n_{2,\lambda}\cos\left(\theta_t\right)}{n_{1,\lambda}\cos\left(\theta_i\right) + n_{2,\lambda}\cos\left(\theta_t\right)}\right|^2 , \tag{13}$$

$$R_{P,\lambda} = \left|\frac{n_{1,\lambda}\cos\left(\theta_t\right) - n_{2,\lambda}\cos\left(\theta_i\right)}{n_{1,\lambda}\cos\left(\theta_t\right) + n_{2,\lambda}\cos\left(\theta_i\right)}\right|^2 . \tag{14}$$

Celkovou reflektivitu pak získáme jednoduše jako aritmetický průměr těchto dvou hodnot

$$R_\lambda = \frac{R_{S,\lambda} + R_{P,\lambda}}{2} . \tag{15}$$

Transmisi pak získáme odečtením reflektivity od jedničky, čili

$$T_\lambda = 1 - R_\lambda . \tag{16}$$

Tím bude dodržen zákon zachování energie. Tyto rovnice platí v případech pro rozhraní dvou dielektrických prostředí. V případě, že jedním z rozhraní je materiál pohlcující

světlo, jako jsou kovy nebo polovodiče, které mají velký absorpční koeficient, jsou rovnice odlišné

$$R_{S,\lambda} = \left| \frac{\left(n_\lambda^2 + k_\lambda^2\right) - 2n_\lambda \cos\left(\theta_i\right) + \cos\left(\theta_i\right)^2}{\left(n_\lambda^2 + k_\lambda^2\right) + 2n_\lambda \cos\left(\theta_i\right) + \cos\left(\theta_i\right)^2} \right|^2 , \tag{17}$$

$$R_{P,\lambda} = \left| \frac{\left(n_\lambda^2 + k_\lambda^2\right) \cos\left(\theta_i\right)^2 - 2n_\lambda \cos\left(\theta_i\right) + 1}{\left(n_\lambda^2 + k_\lambda^2\right) \cos\left(\theta_i\right)^2 + 2n_\lambda \cos\left(\theta_i\right) + 1} \right|^2 . \tag{18}$$

Hodnota n_λ je zde index lomu materiálu, do kterého paprsek vstupuje a k_λ představuje koeficient útlumu materiálu. Celková hodnota reflexivity se pak počítá stejně. Transmise se v tomto případě většinou neuvažuje, jelikož je téměř nulová.

3.6 Beer-Lambertův zákon

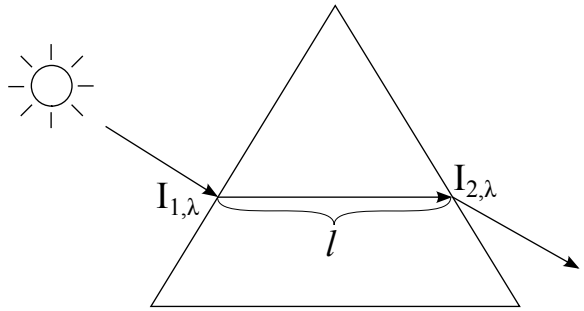
Beer-Lambertův zákon popisuje útlum světla, ke kterému dochází při jeho průchodu dielektrickým prostředím. Absorbance se také někdy nazývá atenuací, která vyjadřuje schopnost paprsku světla proniknout daným optickým prostředím. Díky tomuto jevu pak dochází k obarvení různých dielektrických materiálů, jako jsou například drahokamy. Hodnota absorbance A_λ pro danou vlnovou délku je určena jako

$$A_\lambda = e^{-l\alpha_\lambda} , \tag{19}$$

konstanta l je vzdálenost, kterou světlo v daném prostředí urazí a α_λ je absorpční koeficient. Ten určíme vztahem

$$\alpha_\lambda = \frac{4\pi k_\lambda}{\lambda} . \tag{20}$$

Jako u Fresnelových rovnic, hodnota k představuje komplexní část indexu lomu materiálu, neboli koeficient útlumu. λ pak představuje patřičnou vlnovou délku. Důležité je uvědomit si, že absorpční koeficient α a vzdálenost l jsou řádově rozlišné, proto je potřeba při jejich použití je převést do stejného řádu.



Obrázek 10: Ukázka průchodu paprsku prostředím

Na obrázku 10 je znázorněn průchod paprsku prostředím tělesa. Vstupní intenzita světla je $I_{1,\lambda}$. Intenzita po průchodu tělesem je $I_{2,\lambda} = A_\lambda I_{2,\lambda}$. Absorbance je tedy koeficient, který určuje jak velká část světla pronikne danou vzdáleností v daném prostředí.

3.7 Akcelerační struktury

Při trasování potřebujeme pro každý paprsek zjistit, který nejbližší polygon ve scéně protne. Ověřujeme tedy jeho průsečík se všemi polygony. Tento přístup je však značně neefektivní, protože velká část polygonů je zpravidla mimo dráhu paprsku. Proto je vhodné vybrat z nich jen ty, které paprsku stojí v cestě nebo jsou v její blízkosti. Jak toho ale dosáhnout? S použitím akcelerační struktury. Ta nám umožňuje zjistit, kde v prostoru se paprsek pohybuje a tím i určit, které polygony může potencionálně protnout.

Princip fungování je celkem jednoduchý. Akcelerační struktura je totiž hierarchie, ve které je uchovávána dekompozice prostoru scény na menší podprostory, u nichž si zároveň pamatujeme, které polygony v nich leží. Tyto podprostory pak postupně tvoří stromovou strukturu. Uvážíme-li, že víme, kde ve scéně je počátek paprsku a kterým směrem se bude pohybovat, můžeme vybrat pouze ty podprostory, které při své cestě může navštívit a tím i značně omezit počet polygonů, u nichž je třeba zkontrolovat, zda je paprsek protne či nikoliv.

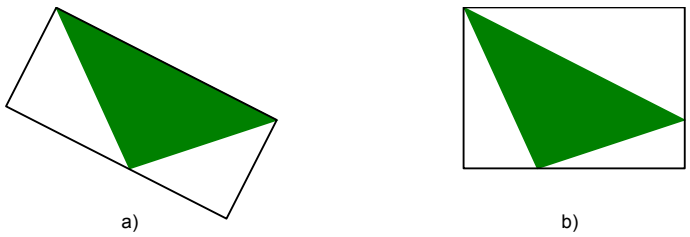
Z hlediska implementace se jedná o stromovou strukturu. V každém uzlu je uchovávána informace o obalové struktuře a indexy polygonů, které obaluje. Každý tento uzel pak má dva či více potomků, v závislosti na druhu akcelerační struktury. Při průchodu touto strukturou, čemuž se také říká traverzace, se začíná v kořenovém uzlu. Nejprve se spočítá, zda paprsek při své cestě protne podprostor daného uzlu. V případě, že ano a uzel není listem, se pokračuje průsečíky s potomky aktuálního uzlu. Je-li uzel zároveň listem, počítají se průsečíky s polygony, které jsou k danému uzlu přiřazeny. Takhle se projde celá struktura, což nám zajistí, že žádný polygon, který by mohl paprsek protnout, nebude vynechán.

3.7.1 Obalové struktury

Obalové struktury (bounding box) mají za úkol vymezit podprostor patřičné velikosti. Typickým tvarem používaným pro tyto obaly jsou kvádry (proto box). Často se používají například pro zjednodušení geometrií při kolizních testech ve fyzikálních simulacích. Toto využití nás však nezajímá. V případě akcelerace slouží k vymezení prostoru, ve kterém se nachází polygony ve scéně, případně nějaká jejich část - podprostor. Ačkoliv jsou tyto obalové struktury zpravidla obecné a mohou být v prostoru libovolně rotovány, při konstrukci akceleračních struktur se nejčastěji používají osově zarovnané obalové struktury - Axis Aligned Bounding Box (AABB). Jejich osy jsou zarovnány s osami souřadného systému prostoru a nelze s nimi rotovat. To značně ulehčuje jak implementaci jejich vytvoření, tak i výpočet průsečíku paprsku s nimi.

Programově je dostačující u této struktury uchovávat vektory, které představují rozměr obalu. Tento rozměr je uložen jako minimální a maximální souřadnice ve všech osách souřadného systému. Například, pro trojúhelník definovaný body $A\,[0,1,4]$, $B\,[2,3,-1]$ a $C\,[1,-1,2]$ bude minimální rozsah $MIN\,[0,-1,-1]$ a maximální $MAX\,[2,3,4]$. Pomocí znalosti těchto rozsahů pak jsme schopni sestavit patřičný kvádr a spočítat s ním průsečík.

Další důležitou vlastností je slučování dvou obalů do jednoho. Máme-li dvě různé obalové struktury, můžeme je sloučit, čímž vznikne nový a větší obal. Uvažujme obal



Obrázek 11: Ukázka obalových struktur: a) obecný obal, b) AABB

$A = MIN\left[0,-1,-1\right], MAX\left[2,3,4\right]$ a druhý obal $B = MIN\left[-3,5,3\right], MAX\left[1,8,5\right]$, vznikne jejich sloučením obal $C = MIN\left[-3,-1,-1\right], MAX\left[2,8,5\right]$. Pomocí tohoto slučování obalů jsme kolem jednotlivých trojúhelníků schopni vymezit prostor, ve kterém se nacházejí.

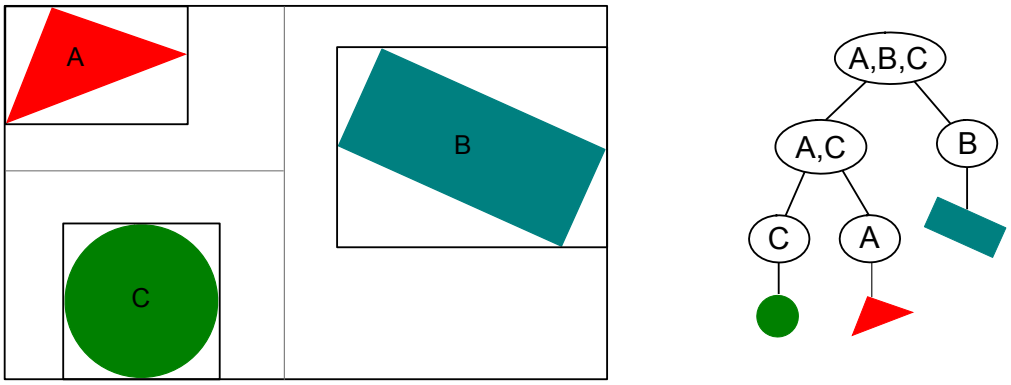
3.7.2 Bounding Volume Hierarchy - BVH

BVH je v současnosti jeden z nejpoužívanějších druhů akcelerační strukutry, mimo jiné i díky výzkumu společnosti NVIDIA, která se snaží tuto strukturu, resp. algoritmy pro její konstrukci, stále zdokonalovat (více v kapitole 4.3).

Jak bylo popsáno výše, BVH je stromová hierarchie podprostorů scény, přesněji se jedná o binární strom. Každý uzel stromu má tedy dva potomky, kteří dále dělí podprostor svého rodiče. Otázkou je, jak toto dělení probíhá, tedy jak je celá hierarchie vytvořena. Prvním krokem je vytvoření kořenu stromu. V něm se nachází všechny trojúhelníky scény a sestaví se pro ně obalová struktura. Poté je potřeba vzniklou oblast dále rozdělit. BVH při své konstrukci oblast rozdělí podle jedné z os prostoru, v prvním kroku například podle osy $\mathbf{x}$. Když je vybrána osa dělení, setřídí se všechny trojúhelníky podle jejich souřadnice na této ose. Jejich interval se pak rozdělí na dvě poloviny, které budou tvořit potomky kořenového uzlu. Pro každého potomka se pak opět sestaví podprostor, který jemu náležící trojúhelníky zaujímají, vybere se jiná osa dělení (zpravidla se postupuje cyklicky přes $\mathbf{x}$, $\mathbf{y}$ a $\mathbf{z}$), opět se trojúhelníky v daném uzlu setřídí. Takto se rekurzivně pokračuje do té doby, dokud není splněno nějaké kritérium pro dělení. Tím je v tomto případě maximální počet trojúhelníků, který může uzlu náležet. Výsledkem tohoto postupu je použitelná, ale ne optimální akcelerační struktura.

Obrázek 12 ukazuje příklad BVH. Vlevo je scéna, nad kterou je struktura vytvořena, vpravo výsledný strom. Kořen obsahuje celou scénu, tedy objekty A, B i C. Prostor je následně rozdělen podle osy $\mathbf{x}$, vytvářející dva potomky. Jeden obsahuje obejkt B a je zároveň listem stromu, druhý objekty A a C. Následně je provedeno dělení levého potomka v ose $\mathbf{y}$, vytvářející pro něj potomky (listy), každý obsahující jeden z objektů podprostoru jejich rodiče.

Surface Area Heuristic (SAH) je pokročilý algoritmus konstrukce BVH, který přináší nové kritérium pro dělení uzlů, díky němuž je kvalita výsledné hierarchie vyšší. Zde se rozhoduje, zda je výpočetně lepší projít uzel ve stavu, v jakém je, nebo jej rozdělit a projít potomky. K tomuto rozhodnutí se využívá dvou konstant a rovnic, které s nimi počítají.



Obrázek 12: Ukázka BVH

Těmito konstantami jsou cena pro průchod uzlem C_t a cena výpočtu průsečíku paprsku s trojúhelníkem C_i. S jejich pomocí jsme schopni rozhodnout, zda má dojít k dělení či nikoliv a podle jaké osy se má případně provést. Cena C_{split} průchodu při uzlem při jeho dělení se vypočítá dle vztahu

$$C_{\text{split}} = C_t + \frac{SA(B_L)}{SA(B)} N_L C_i + \frac{SA(B_R)}{SA(B)} N_R C_i \ , \tag{21}$$

kde $SA(..)$ vypočte povrch obalové struktury, B_L je obal levého potomka, B_R pravého a B rodiče, N_L je počet trojúhelníků levého potomka a N_R počet trojúhelníků pravého potomka. Cena průchodu uzlem bez jeho rozdělení pak jako

$$C_{\text{parent}} = NC_i \ , \tag{22}$$

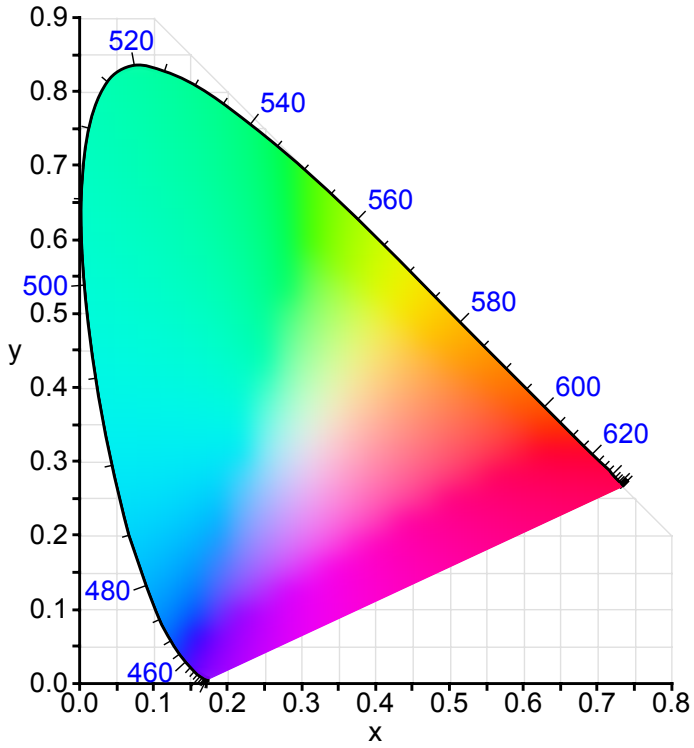
kde N je počet trojúhelníků v uzlu. Pomocí porovnávání těchto dvou cen jsme pak schopni najít nejvhodnější místo pro dělení uzlu. V případě, že C_{parent} je vždy menší než C_{split}, neexistuje vhodné místo pro provedení dělení a uzel je stává listem stromu. Nevýhodou tohoto přístupu je, že konstanty C_t a C_i se musí nastavovat experimentálně a ne pro každou dvojici se strukturu podaří vytvořit.

3.8 Barevné prostory

Než si představíme konkrétní barevné prostory, popíšeme si, co to vlastně barevný prostor je. Základem barevného prostoru je barevný model, který nám dává abstraktní matematický popis, jak lze barvy vyjádřit pomocí n-tic čísel, nejčastěji trojic - viz 3.2.2. Mezi nejznámější barevné modely v dnešní době patří RGB model. Model RGB pracuje se třemi základními barvami: červenou, zelenou a modrou, z nichž se odvíjí i jeho název. Tyto barvy byly zvoleny na základně toho, jak čípky v lidském oku vnímají jednotlivé záření. Zároveň je RGB aditivní barevný model, což znamená, že se jednotlivé barevné složky míchají a výsledkem jsou další barevné odstíny, případně vyšší intenzita barvy.

Když k tomuto modelu definujeme, jak mají být tyto n-tice interpretovány, dostáváme barevný prostor. Barevný prostor je tedy definován rozsahem barev, které dokáže zobrazit.

Tomuto rozsahu se také říká gamut. Ten se zpravidla zobrazuje jako oblast v CIE 1931 chromatickém diagramu.



Obrázek 13: CIE 1931 chromatický diagram, zdroj: *Wikipedia* [10]

3.8.1 CIE 1931 XYZ

Prvním matematicky definovaným barevným prostorem je CIE 1931 XYZ [4]. Tento barevný prostor neobsahuje žádné primární barvy, které by se daly generovat světelným spektrem. Můžeme v něm však vyjádřit veškeré okem viditelné barevné vjemy, které znázorňuje chromatický diagram, viz obrázek 13. Je to díky analogii X, Y a Z na hodnoty S, M a L (viz tristimulus 3.2.2). Y představuje svítivost, Z zhruba odpovídá průběhu S (tedy stimulaci modré barvy) a X je lineární kombinace průběhů S, M a L taková, aby nebyla negativní.

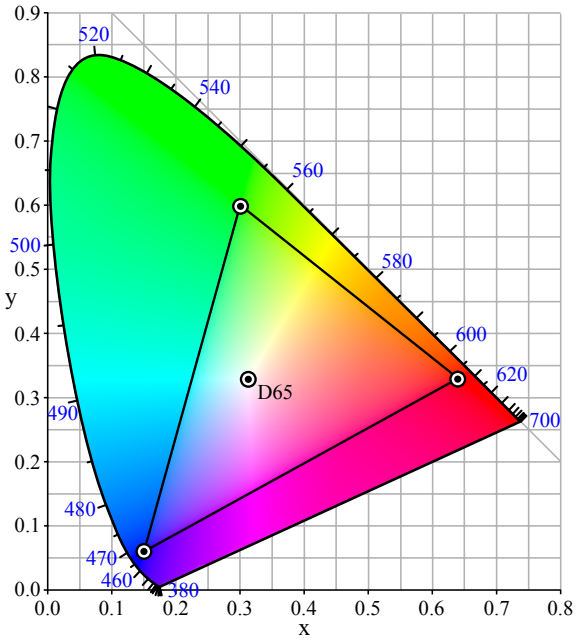
3.8.2 sRGB

Barevný prostor sRGB je založen na RGB barevném modelu. Jedná se tedy o aditivní barevný prostor, kde tristimulus (0, 0, 0) představuje černou barvu a (1, 1, 1) bílou barvu. Primárními barvami jsou červená, zelená a modrá. Složky tristimulu mají stejné pořadí. sRGB také obsahuje nelineární transformaci mezi intenzitou barev a skutečnou uloženou hodnotou. Ta byla navržena tak, aby odrážela gamma křivku reálného monitoru, který

má gamma hodnotu přibližně 2.2. Tento barevný prostor byl navržen pro použití třeba u monitorů a kódování barev na internetu.

Hodnota	Červená	Zelená	Modrá	Bílý bod
x	0.6400	0.3000	0.1500	0.3127
y	0.3300	0.6000	0.0600	0.3290
Y	0.2126	0.7153	0.0721	1.0000

Tabulka 1: Primární barvy sRGB prostoru a vztah k chromatickému diagramu



Obrázek 14: Gamut barevného prostoru sRGB, zdroj: *Wikipedia* [11]

Obrázek znázorňuje gamut tohoto barevného prostoru. Vidíme zde vyznačený bílý bod i všechny tři primární barvy. Jak je patrné, tento barevný prostor nedokáže zobrazit všechny barvy, které je lidské oko schopné vnímat.

3.8.3 Převod spektrálních vzorků do sRGB

Důležitým krokem při trasování je převod spektrálních vzorků tak, abychom je mohli zobrazit. Tento převod se skládá ze 3 kroků. Prvním z nich je převedení spektrálních vzorků do CIE 1931 XYZ barevného prostoru. Toho lze dosáhnout použitím následujících vztahů:

$$X = \frac{1}{N} \int_{380}^{780} S\left(\lambda\right) I\left(\lambda\right) \bar{x}\left(\lambda\right) d\lambda \ , \tag{23}$$

$$Y = \frac{1}{N} \int_{380}^{780} S\left(\lambda\right) I\left(\lambda\right) \bar{y}\left(\lambda\right) d\lambda \ , \tag{24}$$

$$Z = \frac{1}{N} \int_{380}^{780} S\left(\lambda\right) I\left(\lambda\right) \bar{z}\left(\lambda\right) d\lambda \ , \tag{25}$$

$$N = \int_{380}^{780} I\left(\lambda\right) \bar{y}\left(\lambda\right) d\lambda \ , \tag{26}$$

kde $\bar{x}$, $\bar{y}$ a $\bar{z}$ jsou funkce standardního pozorovatele, $I\left(\lambda\right)$ představuje referenční osvětlení a $S\left(\lambda\right)$ je spektrální vzorek, který převádíme. Člen $\frac{1}{N}$ je zde kvůli normalizaci jasu vzhledem k referenčnímu světlu.

Poté následuje převod z CIE 1931 XYZ do sRGB prostoru. Tento převod má dvě fáze. Nejdříve převedeme hodnoty XYZ do lineárního RGB prostoru, ten má lineární gamma křivku a jeho hodnoty se zpravidla označují malými písmeny *r*, *g*, *b*. V tomto případě se jedná o jednoduché násobení matice a vektoru:

$$\begin{bmatrix} r \\ g \\ b \end{bmatrix} = [M]^{-1} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \ , \tag{27}$$

hodnoty pro matici $[M]^{-1}$ lze nalézt například na *www.brucelindbloom.com* [5]. Druhou a poslední fází je gamma korekce. Nechť C_{linear} představuje lineární RGB hodnoty a C_{srgb} hodnoty v sRGB prostoru, pak:

$$C_{\mathrm{srgb}} = \begin{cases} 12.96\, C_{\mathrm{linear}} \ , & C_{\mathrm{linear}} \leq 0.0031308 \\ 1.055\, C_{\mathrm{linear}}^{1/2.4} - 0.055 \ , & C_{\mathrm{linear}} > 0.0031308 \end{cases} . \tag{28}$$

Tento vztah se aplikuje na každý barevný kanál samostatně. Nyní již máme výslednou barvu C_{srgb} připravenou k zobrazení.

4 NVIDIA OptiX SDK

OptiX je framework společnosti NVIDIA využívající architekturu CUDA, který přenáší ray tracing na grafické akcelerátory. Díky tomu je použití fotorealistických zobrazovacích metod mnohem rychlejší a jednodušší. OptiX je zároveň velmi flexibilní. Není totiž omezen vestavěnými strukturami pro uživatelská data, čímž umožňuje využití i v oblastech mimo renderování.

Díky své architektuře umožňuje vývojářům rychle sestavit software využívající ray tracing tak, jak to potřebují. Navíc se odpadá nutnost učit se nový specifický programovací jazyk. OptiX totiž poskytuje API pro jazyk C/C++. Funkce, které se vykonávají na GPU, což jsou například funkce pro výpočty osvětlení, se pišou v jazyce CUDA C/C++. Proto je integrace OptiX SDK velmi jednoduchá.

OptiX dále nabízí integrovanou interoperabilitu s OpenGL a Direct3D. To umožňuje jednoduše zobrazit výsledek trasování, aniž by byla potřeba kopírovat data do paměti hostitele. Samozřejmostí je spolupráce s CUDA kernely.

Licence pro použití byla do nedávna zdarma pro jakékoliv účely, tj. i pro komerční využití. To se však změnilo s příchodem verze 3.5, která přináší komerční licenci. Zároveň však poskytuje další funkčnost a ukázkové implementace, spolu s lepší podporou, kterou komerční projekty vyžadují.

4.1 Využití OptiX SDK

Co se využití týče, nabízí se celá řada oblastí. Mimo zobrazovací úlohy jsou to třeba optické/akustické návrhy nebo šíření záření (například teplo při zkoumání chlazení), jednoduše všude tam, kde se k řešení využívá sledování paprsků. To je umožněno tím, že každý paprsek v OptiX může nést libovolnou datovou strukturu a tak při své cestě scénou zaznamenávat libovolná data.

Pro účely zrychlení vykreslení jej využívá například filmové studio Pixar. Před integrací OptiX do produkce museli spoustu efektů ve scénách simulovat uměle, což vedlo k vysokému počtu světel (více než 100). OptiX jim umožnil ve filmu *Univerzita pro příšerky* nasvítit scénu pouze 12 světly, přičemž kvalita výsledku byla stejná a čas výpočtu kratší.

Další zajímavou oblastí je interaktivní ray tracing, tedy ray tracing v reálném čase. Díky vysoké optimalizaci frameworku, neustálému vývoji a stále rostoucímu výpočetnímu výkonu grafických akcelerátorů s technologií CUDA, se otevírá možnost integrace fotorealistického zobrazování například pro použití v počítačových hrách.

4.2 Architektura SDK

Architektura OptiX SDK je ve své podstatě rozdělena do čtyř základních částí, které však v celkovém efektu musí spolupracovat.

- Programy a proměnné - tato část se stará o programy obsahující potřebné výpočty a proměnné, které jsou v nich používány.

- Geometrie - slouží k popisu elementárních tvarů, ze kterých se skládá scéna. Drží informace o geometrických výpočtech nad těmito tvary, například o výpočtech průsečíků s paprsky.
- Stavba scény - nabízí prvky, které umožňují sestavit hierarchii objektů ve scéně a akcelerační struktury přesně podle potřeb implementované aplikace.
- Data - práce s buffery a texturami.

Podrobný popis lze najít v textu [6] a dokumentaci dodávané s SDK (doporučuji projít OptiX Programming Guide). V následujícím textu se zaměříme pouze na části použité v implementaci. Pro jednoduchost nebudou uvedeny parametry a návratové hodnoty jednotlivých funkcí.

4.2.1 Kontext

Základním prvkem je třída `RTcontext`. Ten se vytváří funkcí `rtContextCreate`. Tato třída má za úkol držet veškeré informace potřebné pro trasování. Zároveň se nastavuje několik důležitých parametrů.

- Velikost zásobníku - `rtContextSetStackSize` - nastavuje velikost zásobníku pro každé vlákno, které v rámci kontextu poběží. Tato hodnota je důležitá, protože má přímý vliv na to, jak hluboká může být rekurze. Je-li zásobník příliš malý, dojde při zanořování k jeho přetečení a trasování selže. Naopak je-li příliš vysoká, nebude kvůli nedostatku paměti spuštěn maximální možný počet vláken. Nastavení je tedy potřeba zjistit experimentálně.
- Počet druhů paprsků - `rtContextSetRayTypeCount` - jak název napovídá, jedná se o to, kolik různých druhů paprsků se v kontextu bude používat. Díky tomu můžeme rozlišit paprsky v rámci použití (například pro osvětlení a kontrolu zastínění).
- Počet vstupních bodů - `rtContextSetEntryPointCount` - vstupní bod je program, kterým začíná výpočet pixelu. Zpravidla zde dochází ke generování paprsků. Můžeme tedy jednoduše spouštět různé druhy výpočtů nad daty podle potřeby.

V okamžiku, kdy jsou všechny data kontextu nastavena, jej můžeme spustit. Nejdříve je však potřeba provést validaci pomocí `rtContextValidate`. Poté můžeme kontext zkompilovat přes `rtContextCompile`. Nyní můžeme provést samotné spuštění: `rtContextLaunchND`, kde místo `N` je dimenze spuštění - 1, 2 nebo 3. Nakonec, když už kontext není potřebný, se zničí funkcí `rtContextDestroy`. Tím dojde ke správnému uvolnění zdrojů, které si kontext alokoval.

4.2.2 Programy

Další podstatnou část tvoří programy `RTprogram`. Ty se vytvářejí z .PTX souborů pomocí funkce `rtProgramCreateFromPTXFile`. Programy mají různé účely, které si blíže popíšeme.

- Ray Generation - `rtContextSetRayGenerationProgram` - tento program se navazuje na vstupní bod kontextu. V rámci ray tracingu jeho implementace definuje, o jaký druh kamery se bude jednat. Můžeme zde řešit supersampling, adaptivní rozlišení pro interaktivní implementace a další.
- Intersection - `rtGeometrySetIntersectionProgram` - program obsluhuje výpočet paprsku s primitivem. Zároveň se zde mohou počítat normály, texturovací souřadnice a jakákoliv další logika, kterou aplikace v tomto okamžiku potřebuje. V rámci tohoto programu se, mimo jiné, mohou provádět dvě specifická volání. První z nich je `rtPotentialIntersection`. Ta ověří, zda je daný průsečík bližší než ten předchozí. Druhé volání je `rtReportIntersection`. Tato funkce potvrzuje průsečík a v parametru předává index materiálu, který se má postarat o patřičné výpočty.
- Bounding Box - `rtGeometrySetBoundingBoxProgram` - obsluhuje vytvoření obalové struktury při konstrukci akcelerační struktury.
- Closest Hit - `rtMaterialSetClosestHitProgram` - program obsluhující výpočet pro nejbližší nalezený průsečík daného paprsku. Může například vysílat další paprsky do scény a ukládat výsledky do patřičných struktur.
- Any Hit - `rtMaterialSetAnyHitProgram` - tento program je volán při každém nalezení průsečíků. Zpravidla se využívá při zjišťování, zda na určitý bod dopadá světlo.
- Miss - `rtContextSetMissProgram` - tento program se vyvolá v okamžiku, kdy paprsek nemá žádný průsečík se scénou. Můžeme zde například nastavit konstantní barvu pozadí nebo vzorkovat environmentální mapu.
- Exception - `rtContextSetExceptionProgram` - program sloužící pro obsluhu výjimek, který je vázán na konkrétní vstupní bod. Můžeme tedy definovat různé chování pro různé vstupní body.

Všechny tyto programy se píšou jako kernely v CUDA C/C++ jazyce, načež se kompilují do .PTX souborů. Jedinou změnou je zde nahrazení druhu CUDA kernelu, místo `__global__` se používá `RT_PROGRAM`, což je jen předefinované `__global__`. Kernely navíc nesmí mít návratovou hodnotu, musí tedy být `void`. Název kernelu je na vývojáři, při načítání programů z .PTX souborů se tento název uvádí, aby OptiX poznal, který kernel ze souboru má načíst. Vstupní parametry závisí na druhu programu, viz dokumentace SDK. Například closest hit a any hit programy nemají žádné vstupní parametry. Také je dobré poznamenat, že podobně jako u GLSL je možné mezi intersection a closest/any hit programy předávat proměnné pomocí atributů.

4.2.3 Proměnné

Další nedílnou součást tvoří proměnné. Ty se dělí na 2 druhy. Jsou buďto deklarovány v rámci kontextu nebo pro určitý program.

Kontextové proměnné se deklarují voláním `rtContextDeclareVariable`. Tyto proměnné jsou pak přístupné ve všech programech. Hodí se tedy pro předávání globálních nastavení. K nastavení hodnoty pak slouží `rtVariableSetN`. `N` určuje jaká data se proměnné přiřazují. `rtVariableSet3f` tedy nastavuje třísložkový vektor hodnot s plovoucí desetinnou čárkou.

Proměnné pro program se definují funkcí `rtProgramDeclareVariable`. Díky nim lze například mezi snímky při interaktivním trasování měnit polohu a orientaci kamery.

V zdrojových souborech s kernely se proměnné deklarují přes `rtDeclareVariable`. Přes tyto proměnné je možno využít pro přístup k interním proměnným OptiX, jako jsou například současný paprsek nebo velikost spuštěného kontextu (rozměry spuštění). Také, jak bylo zmíněno výše, jejich pomocí předávat atributy mezi jednotlivými programy. V neposlední řadě slouží i k přejímání dat. Zároveň musí všechny proměnné deklarované na straně hostitele (CPU), ať už kontextové nebo pro program, mít svůj protipól deklarovaný voláním `rtDeclareVariable`.

4.2.4 Buffery

Buffery slouží k ukládání dat a vytváří se funkcí `rtBufferCreate`. Jedním z parametrů této funkce je, jak se do bufferu bude přistupovat. Možnostmi jsou `RT_BUFFER_INPUT`, `RT_BUFFER_OUTPUT` nebo `RT_BUFFER_INPUT_OUTPUT`. `RT_BUFFER_INPUT` říká, že hostitel do něj může pouze zapisovat a karta pouze číst. `RT_BUFFER_OUTPUT` je pravý opak, tedy hostitel pouze čte a karta pouze zapisuje. Poslední možnost umožňuje oběma stranám zápis i čtení. Když je buffer vytvořen, musí se specifikovat jeho formát, tedy jaký datový typ bude mít jeden prvek bufferu. K tomu slouží funkce `rtBufferSetFormat`. Zde je možné použít buďto předem definované datové typy, jako `RT_FORMAT_FLOAT`, nebo vlastní datovou strukturu pomocí `RT_FORMAT_USER`. V případě uživatelského formátu je zároveň nutné definovat, kolik bytů jeden prvek zabírá. K tomu slouží funkce `rtBufferSetElementSize`. Je nutné si uvědomit, že architektura CUDA pracuje s daty zarovnanými na 16 bytů, tedy i velikost uživatelské struktury musí být zarovnaná na 16 bytů. Také je potřeba definovat rozměry bufferu, tedy kolik prvků bude obsahovat. Toho lze dosáhnout funkcí `rtBufferSetSizeND`, kde jako u `rtContextLaunchND` konstanta `N` definuje rozměr - 1, 2 nebo 3.

Když je buffer nastaven, můžeme jej v případě potřeby (tedy u bufferů, které umožňují zápis hostitele) naplnit daty. Nejdříve si jej funkcí `rtBufferMap` navážeme na `void` ukazatel v paměti hostitele. Poté na tento ukazatel nakopírujeme požadovaná data a následně jej funkcí `rtBufferUnmap` opět uvolníme. Nakonec si voláním `rtBufferValidate` ověříme, zda spolu jednotlivá nastavení nekolidují.

4.2.5 Paprsky

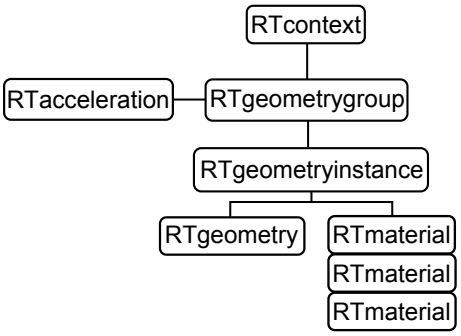
OptiX SDK má vnitřně definovanou strukturu paprsku. Ta obsahuje počátek a směr paprsku, index typu paprsku a minimální/maximální vzdálenost, kterou může ve scéně urazit. Paprsky lze vytvářet buďto pomocí konstruktorů nebo funkce `make_Ray`. Při jejich tvorbě se určují všechny parametry, které struktura obsahuje.

Jak bylo napsáno výše, OptiX neomezuje vývojáře v tom, jakou datovou strukturu si s sebou paprsek může nést. Tato struktura se také označuje termínem payload. Díky tomu můžeme při trasování počítat libovolná data.

OptiX dále umožňuje paprsek, resp. větev paprsků v případě potřeby ukončit funkcí `rtTerminateRay`. V případech, kdy při nalezení průsečíku chceme provést nějakou operaci a zároveň se zde paprsek nemá zastavit, ale pokračovat dál, použijeme funkci `rtIgnoreIntersection`. Toto se hodí například při výpočtu zastínění bodu průhledným tělesem. Tyto volání však nelze provádět v libovolném programu, ale jsou zde jistá omezení. Zde bych čtenáře opět odkázal na podrobnou dokumentaci.

4.2.6 Geometrie a akcelerace

Třídy pro reprezentaci grafu scény jsou sestaveny ve stromové hierarchii. Tuto hierarchii znázorňuje následující obrázek.



Obrázek 15: Ukázka hierarchie pro správu geometrií v OptiX SDK

Základem je třída `RTgeometry`. Ta definuje, z jakých primitivních objektů je scéna složena. U této třídy se definují intersection a bounding box programy a počet primitiv.

Na stejné úrovni je `RTmaterial`. Tato třída je jakási kolekce closest a any hit programů pro každý druh materiálu. Když se tedy v intersection programu nahlásí průsečík, indexem se vybere materiál, který pak podle druhu paprsku aktivuje patřičný program, který se o něj má postarat. Výhodou je, že u paprsku nemusí být specifikovány oba druhy programů, tedy closest a any hit. Je však důležité si pohlídat, aby nedošlo k volání nepřiřazeného programu. V tom případě by trasování selhalo.

Nad nimi je postavena třída `RTgeometryinstance`. Zde se přiřazuje instance třídy `RTgeometry`, funkcí `rtGeometryInstanceSetMaterialCount` se určí počet materiálů, které se starají o paprsky a následně se pomocí `rtGeometryInstanceSetMaterial` na jednotlivé indexy přiřadí konkrétní materiály.

Informace o akcelerační struktuře obsluhuje třída `RTacceleration`. Pro tu existují různé druhy konstrukce a průchodu. Jejich seznam je obsažen v dokumentaci. Druh konstrukce se specifikuje přes `rtAccelerationSetBuilder`, průchod pomocí `rtAccelerationSetTraverser`. U akcelerační struktury je zároveň velmi důležité nastavit jí ukazatel na vertex a index buffery pomocí `rtAccelerationSetProperty`.

U těchto bufferů je zároveň potřeba definovat, kolik dat je mezi jednotlivými pozicemi, resp. trojúhelníky. Toto je však nutné pouze v případě, že pozice vertexů nebo indexy jsou prokládány jinými informacemi. Jsou-li nahuštěny těsně za sebou, není toto nastavení vyžadováno.

Třída `RTgeometrygroup` představuje kontejner pro libovolný počet geometry instancí. Patří mezi tzv. "top level"uzly v hierarchii a je tedy nad ní možné spustit trasování. To je mimo jiné i proto, že musí mít přiřazenu akcelerační strukturu funkcí `rtGeometryGroupSetAcceleration`. Zároveň se jí určí, kolik geometry instancí obstarává `rtGeometryGroupSetChildCount` a jednotlivé instance se přiřadí přes funkci `rtGeometryGroupSetChild`.

Ačkoliv v této hierarchii existují i výše položený uzel `RTgroup`, v Performance Guidelines z Optix Programming Guide je přímo uvedeno, že se doporučuje dělat hierarchii co nejmenší.

4.3 Akcelerační struktura

Jednou ze zajímavých vlastností OptiX SDK je algoritmus konstrukce akcelerační struktury. Touto strukturou je SBVH, neboli Split BVH [7]. Motivací výzkumu bylo odstranit vzájemné překrývání obalových struktur jednotlivých uzlů stromu tak, aby nedocházelo k nepotřebným testům pro ray/triangle průsečíky. Tyto nadbytečné testy byly způsobeny tím, že paprsek se trefil do dvou sousedících podprostorů a proto bylo třeba porovnat všechny trojúhelníky v obou z nich. Výsledek konstrukce SBVH minimalizuje překrývání jednotlivých podprostorů a díky tomu paprsek nakonec trefí pouze jednoho ze dvou potomků uzlu. Cenou za kvalitu výsledné struktury je však malá rychlost konstrukce. Nehodí se tedy pro dynamicky se měnící scény.

Ve verzi SDK 3.5 je dostupný nový algoritmus nazvaný Treelet Reordering BVH - TRBVH [8]. Cílem tohoto vývoje bylo poskytnout algoritmus, který by co nejvíce zachoval kvalitu SBVH a zároveň byla jeho konstrukce co nejrychlejší. Toho je dosaženo pomocí úprav stromu.

Jelikož jsou oba přístupy komplikovanější, doporučuji čtenáři v případě zájmu projít patřičné texty, v nichž jsou algoritmy popsány. Nerad bych zde uváděl nepřesné informace, protože jsem se jimi hlouběji nezabýval.

5 Implementace

V této kapitole se podíváme na praktickou implementaci spektrálního sledování paprsků. V práci jsem vycházel z výše popsaného Whittedova rekurzivního algoritmu. Potřebné úpravy nebyly příliš rozsáhlé. Bylo nutné algoritmus pouze přepsat tak, aby pro každý pixel vypočítal všech 81 spektrálních vzorků, které se před zobrazením převedly do sRGB. V následující kapitole se však nebudeme zabývat tím, jak načítat data scény ani implementací samotného Whittedova trasování. Budou popsány pouze části implementace, které se spektrálním trasováním přímo souvisí, je tedy vhodné, aby si čtenář, v případě potřeby, zopakoval, jak se Whittedův algoritmus implementuje.

Nejdříve se však podívejme, jak lze k spektrálnímu trasování programově přistoupit. V podstatě existují 3 možnosti, jak lze implementovat průchod spektrální scénou. Ty se liší v druhu použitých paprsků.

První možnost je použití *monochromatických paprsků*. Jak název napovídá, tyto paprsky reprezentují pouze jednu barvu, čili vlnovou délku. Výhodou tohoto přístupu je jednodušší implementace. Programátor pak u každého paprsku pouze hlídá příslušnou vlnovou délku a podle ní přistupuje k informacím v materiálech a světlech, což přináší i možnost kód lépe optimalizovat. Nevýhoda tohoto přístupu však je, že už na počátku trasování pixelu je vysláno 81 paprsků pro každý vzorek spektra a pro každý z nich se musí akcelerační struktura procházet zvlášť. Toto zvyšuje výpočetní nároky. Také trasování jednotlivých spektrálních vzorků potřebujeme až případě, že dojde k disperzi světla. Když scéna žádný disperzní materiál neobsahuje, je tento přístup zbytečný.

Druhá možnost jsou *polychromatické paprsky*. Tyto paprsky již reprezentují celé trasované spektrum, čímž nás zbavují nevýhody nadbytečného procházení BVH z prvního přístupu. Na počátku se vyšle jeden paprsek, takže akcelerační strukturu procházíme pouze jednou. Je však důležité správně implementovat možnost disperze světla. Toto je nevýhoda použití těchto paprsků. V případě disperze jsme zpět odkázáni na trasování každého vzorku samostatně, takže opět potřebujeme monochromatické paprsky.

Tyto dva přístupy však prakticky neumožňují použití difuzních textur. Možnost, jak jejich použití dosáhnout, je trasovat scénu ve dvou průchodech. Samostatně pro spektrální vzorky a pro texturování, načež se oba obrazy vhodně zkombinují. Nebo použít třetí přístup k spektrálnímu trasování. Tím je využití obvyklých *RGB paprsků*. Tyto paprsky jsou stejné jako u běžného trasování, tedy počítají přímo 3 barevné kanály. Díky tomu můžeme do trasování snadno integrovat texturování. Velkou nevýhodou však je, že v každém místě dopadu je třeba spektrální zastoupení barvy převádět na RGB, což jsou zbytečné přepočty. Barvu nám stačí převést až na konci trasování. Navíc nastává problém při disperzi, kdy se opět nevyhneme monochromatickému přístupu.

Z předchozího textu vyplývá, že ideální přístup je použít hybridní implementaci pomocí polychromatických a monochromatických paprsků. V mé implementaci jsem však použil první přístup, tedy čistě použití monochromatických paprsků.

5.1 Datové struktury pro spektrální data

Prvním krokem při implementaci byl návrh vhodných datových struktur pro ukládání spektrálních dat. Je potřeba držet informace o spektrálním rozložení pro jednotlivé barvy, které se pak využívají pro ukládání vlastností materiálů, světel nebo pro převod spektrálních dat do sRGB barevného prostoru.

5.1.1 ColorData

Tato struktura má za úkol držet informace o spektrálním rozložení barvy, tj. ukládá 81 vzorků světla přes viditelné spektrum.

```
struct ColorData
{
  float data_[81];

  ColorData( void )
  {
    for ( int i = 0; i < 81; i++ )
    {
      data_[i] = 0.0f;
    }
  }
};
```

Výpis 3: Struktura ColorData

Struktura také obsahuje konstruktor, který inicializuje pole hodnot `data_` na nuly, aby nebylo potřeba ručně tuto inicializaci provádět před každým použitím.

5.1.2 Material

Struktura materiálu potřebuje držet informace o tom, jakou distribuční funkci materiál zastává, jeho barvu a index lomu. K držení informace o přidružené distribuční funkci jsem pro přehlednost navrhl vlastní enumerátor eBRDF.

```
enum eBRDF
{
  eDIFFUSE = 0,
  ePHONG,
  eDIELECTRIC,
  eCONDUCTOR,
  eMIRROR,
};
```

Výpis 4: Enumerátor eBRDF

Implementace tohoto enumerátoru umožňuje jednoduše přidávat další distribuční funkce do programu, aniž by bylo třeba výrazně zkoumat samotnou strukturu programu. Samotná struktura `Material` je vcelku jednoduchá.

```
struct MaterialData
{
  eBRDF brdf_type_;   // Type of BRDF of material
  ColorData color_;   // Material color
  ColorData n_data_;  // Index of refraction
  ColorData k_data_;  // Absorbtion coefficient
};
```

Výpis 5: Struktura Material

Jak je vidět, obsahuje informaci o distribuční funkci, a spektrální data o barvě, indexu lomu a koeficientu útlumu.

5.1.3 Light

V programu existují dva druhy světel. Jsou jimi bodová a plošná světla. Bodové světlo je ukázáno v následujícím kódu

```
struct PointLight
{
  optix :: float3  position_;  // Light position
  ColorData color_;            // Light color
};
```

Výpis 6: Struktura PointLight

Jak je patrné, bodovému světlu stačí pouze jeho barva a pozice v prostoru. Obecná plošná světla jsou zjednodušena do trojúhelníkových, která umožňují přímější práci při zjišťování, zda došlo k jejich zasažení paprskem. K tomu je potřeba znát pozice jednotlivých vrcholů trojúhelníku a jeho normálu, podle které zjistíme, zda jsme jej trefili z přední či zadní strany.

```
struct TriangleLight
{
  optix :: float3  v1_, v2_, v3_;  // Light vertices
  optix :: float3  normal_;        // Light normal
  ColorData color_;                // Light color
};
```

Výpis 7: Struktura TriangleLight

Tento druh světel je potřebný v případě implementace path tracingu a v mé implementaci nebyl prakticky použit.

5.2 Akcelerační struktura

Pro akcelerační struktury jsem implementoval reprezentace obalové struktury, uzlu stromu a samotné akcelerační struktury.

Obalová struktura je popsána strukturou `AABB`, která je vcelku primitivní. Uchovává pouze rozměr obalu a obsahuje metodu pro sloučení s jiným obalem:

```
struct AABB
{
  Vec3 bounds_[2];

  void Merge( AABB & target );
};
```

Výpis 8: Struktura pro reprezentaci obalové struktury

Uzel stromu je implementován ve struktuře `BVHNode`:

```
struct BVHNode
{
  AABB bounds_;
  int  span_[2];
  BVHNode * child_nodes_[2];

  BVHNode( void );

  BVHNode( int from, int to );

  bool IsLeaf( void );

  ~BVHNode();
};
```

Výpis 9: Struktura pro reprezentaci obaluzlu hierarchie

Význam jednotlivých vlastností je vcelku jasný. `bounds_` uchovává obalovou strukturu daného uzlu a tedy i podprostor, který uzel představuje. `span_` je rozsah indexů trojúhelníků, které do daného uzlu náleží, uloženy jako "od - do". `child_nodes_` ukazují na potomky daného uzlu.

Akcelerační struktura je implementována třídou`BVH`, přičemž využívá obou výše zmíněných struktur. Tato třída již zastupuje konkrétní stromovou hierarchii a obsahuje obslužné metody. Deklaraci třídy lze najít v příloze A. Tato třída obsahuje metody pro obyčejnou konstrukci, konstrukci s použitím SAH, průchod stromem a také výpočty osvětlení pro různé BRDF.

Pro vytvoření hierarchie se zavolá konstruktor, do nějž vstupují datové buffery, nastavení maximální hloubky zanoření paprsku a maximální počet polygonů v listech stromu. Tento konstruktor následně volá metodu `BuildTree`, která z daného rozsahu polygonů vytvoří uzel a podle potřeby jej podle zadané osy rekurzivně rozdělí na dva potomky. Pro třídění polygonů je implementován algoritmus quicksort v metodě `Sort`.

Metoda `BuildSAH` umožňuje sestrojení stromu s využitím surface area heuristiky. Tento přístup využívá metod `CalculateSA` a `CalculateISA` pro výpočet povrchu obalových struktur, resp. jejich inverzních hodnot (v rámci vyhnutí se operátoru dělení). V praxi se mi bohužel nepodařilo najít hodnoty pro C_t a C_i takové, aby konstrukce řádně fungovala. Jelikož však CPU implementace nebyla klíčovou částí práce, rozhodl jsem se tím příliš neztrácet čas.

Průchod hierarchií se spouští funkcí `SpectralTrace`, která pro zadaný paprsek zjistí, kterými uzly stromu projde a tím, které podprostory scény tento paprsek protne. Při průchodu stromem se využívá metoda `RayBoxIntersection` pro zjišťování, zda paprsek prostor daného uzlu protne či nikoliv. V případě že ne, průchod dané větve končí. Když ano, pokračuje se v obou potomcích stromu. Jakmile pří průchodu paprsek dospěje do listu, pomocí metody `RayTriIntersection` se zjišťuje průsečík s každým trojúhelníkem, který do daného listu patří. Jakmile je tento průsečík nalezen, zavolá se patřičná metoda pro výpočet osvětlení, v závislosti na druhu distribuční funkce materiálu.

5.3 Výpočty osvětlení

Pro výpočty osvětlení jsem implementoval metody, které řeší jednotlivé BRDF. Parametry těchto funkcí jsou paprsek, bod dopadu a normála. Tyto funkce jsou:

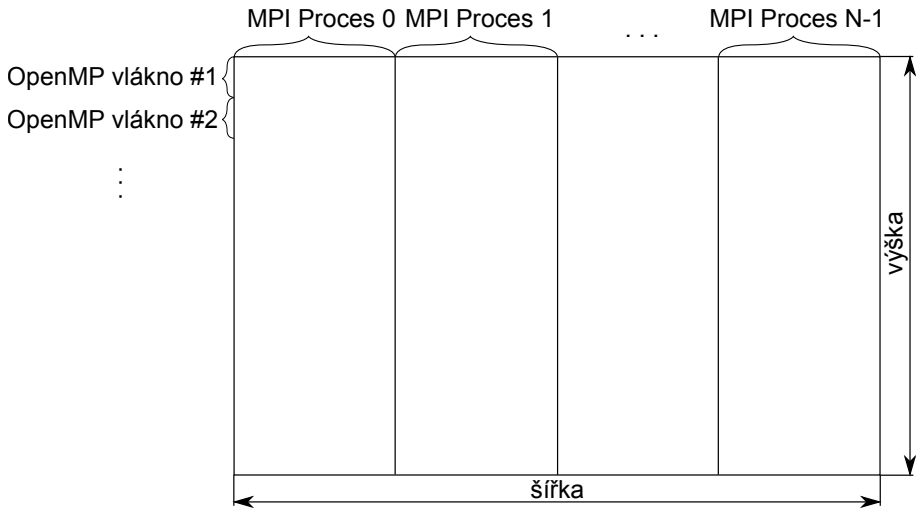
- Diffuse - jak název napovídá, tato funkce počítá difuzní distribuční funkci, tedy funkci, kdy je světlo v místě dopadu šířeno stejně všemi směry.
- Phong - funkce řeší Phongovu BRDF. Vzhledem k absence ostrosti spekulárního odrazu v popisu materiálu je tento koeficient nastaven experimentálně na konstantní hodnotu.
- Mirror - zrcadlová BRDF, dochází pouze k odrazu paprsku, přímé osvětlení v místě dopadu se neřeší.
- Dielectric - dielektrická BRDF, podle Snellova zákonu a Fresnelových rovnich dochází k odrazu a lomu paprsku, kontroluje se i Brewsterův a kritický úhel. Následně se spočítá reflexivita a transmisivita povrchu.
- Conductor - podobně jako Dielectric, ovšem lomený parsek se nepočítá, jeho přínos je nepodstatný. Používají se také alternativní Fresnelovy rovnice pro výpočet reflexivity.

Každá funkce na začátku kontroluje hloubku rekurze. V případě, že hloubka překročí stanovenou hranici, se osvětlení nepočítá. Dále je ve funkcích implementován *importance cutoff*, který zajišťuje, že bude-li příspěvek paprsku ve finální barvě nižší než určitá hranice (experimentálně nastaveno na 0.1%), paprsek se trasovat nebude.

5.4 Paralelizace na CPU

Paralelizace algoritmu na CPU byla provedena s využitím knihoven OpenMP a OpenMPI. Strategie pro paralelizaci byla vcelku přímočará, a to počítat co nejvíce pixelů najednou. Ideálním řešením by bylo obraz rozdělit na části podle toho, jak náročné výpočty v daných oblastech budou. To lze provést pouze v případě, že předem známe vstupní scénu. Toto se však obecně nestává a proto je dobré provést dělení dynamicky.

Knihovna OpenMPI slouží ke spouštění programu ve více procesech a komunikaci mezi nimi. Podle počtu procesů se obraz dynamicky rozdělí na příslušný počet částí.



Obrázek 16: Paralelizace na CPU

Například běží-li šest procesů, je obraz rozdělen na šest částí, kde každý proces dostane za úkol spočítat jednu z nich. Toto dělení obrazu je prováděno na šířku, takže z obrazu o rozlišení 1920x1080 pixelů vznikne šest částí o rozměru 320x1080 pixelů. Každý proces si podle svého ID zjistí, kterou část obrazu má počítat a alokuje si potřebnou paměť. Poté se čeká, než všechny procesy dokončí svou práci pomocí `MPI_Barrier`. V okamžiku, kdy jsou hotovy s výpočty, si master proces (s ID 0) alokuje pole, do kterého uloží celý obraz a pomocí funkce `MPI_Gather` si vyžádá dílčí výsledky od ostatních procesů

```
MPI_Gather( &picture_data, pixels_per_process * 3,  MPI_FLOAT, t, pixels_per_process * 3,
    MPI_FLOAT, 0, MPI_COMM_WORLD );
```

Výpis 10: MPI_Gather pro získání práce všech procesů

Po získání dílčích výsledků je překopíruje do OpenCV obrazu pro zobrazení.

Každá část obrazu v daném procesu je poté rozdělena knihovnou OpenMP na výšku. Možným přístupem k tomuto dělení by opět bylo rozdělit obraz podle počtu vláken. Z obrazu 320x1080 pixelů by při 4 vláknech vznikly části o rozměru 320x270 pixelů. Tato oblast by pak byla přiřazena jednomu vláknu. Toto řešení však není příliš vhodné, protože se může stát, že jedno vlákno narazí na výpočetně náročnější oblast než ostatní a zatímco bude počítat, mohou zbylá vlákna svou práci dokončit a čekat naprázdno. Tím dochází ke zbytečné ztrátě výpočetního výkonu. Mnohem lepší je vláknům přiřazovat menší části obrazu dynamicky. Toho lze docílit v OpenMP následujícím příkazem

```
#pragma omp parallel for schedule( dynamic, chunk_size )
for( int i = 0; i < image_height; i++ )
{
   ...
}
```

Výpis 11: #pragma příkaz pro OpenMP

Tento příkaz rozdělí příslušnou smyčku mezi jednotlivá vlákna. Modifikátor `schedule` kompilátoru říká, jak má být provedeno přiřazování práce vláknům. První parametr určuje druh přiřazování, nejčastěji `static` nebo `dynamic`. Jak je zřejmé, jedná se o statické nebo dynamické přidělení. Druhým parametrem se určuje jak velkou část úkolu má vlákno dostat. Této velikosti se v oblasti rozkládání úloh na menší problémy říká granularita. Její volba je experimentální. Příliš jemná granularita může vést k vysokým nákladům na režii vláken. Hrubá granularita zase může vést k výše zmíněnému problému s uvíznutím vlákna ve výpočetně náročné oblasti. Při mých experimentech se nejvíce vyplatilo přiřazovat vláknům části po 10 řádcích.

Celkově tedy hlavní smyčka programu vypadá následovně:

```
for ( int i = processID * pixel_count; i < ( processID + 1 ) * pixel_count; i++ )
{
  #pragma omp parallel for schedule( dynamic, 10 )
  for ( int j = 0; j < image_height; j++ )
  {
    // Generovani paprsku, spusteni trasovani, ulozeni vysledku
  }
}
```

Výpis 12: Hlavní smyčka programu

Tento postup nám umožňuje pustit kód ve více vláknech na více počítačích. Každý počítač tak dostane za úkol spočítat určitou část obrazu, kterou navíc dále rozdělí mezi daný počet vláken. Na konci výpočtu se částečné výsledky opět odešlou řídícímu procesu, který je složí do výsledného obrazu.

5.5 Převod spektrálních vzorků do RGB

K těmto převodům byla vytvořena třída `ObserverData`, která obsahuje data o standardním pozorovateli a referenčním osvětlení scény.

```
class ObserverData
{
public:
  ObserverData( void );
  ObserverData( char * observer_data, char * reference_white );
  ~ObserverData( void );
  optix :: float3  GetRGB( ColorData * spectral_samples );

private:
  optix :: float3  observer_data_[81];
  ColorData ref_white_;
  float  ref_y_;

  void LoadObserverData( char * file_name );
  void LoadReferenceData( char * file_name );
};
```

Výpis 13: Definice třídy ObserverData

V konstruktoru se pomocí privátních funkcí načtou data standardního pozorovatele a referenčního osvětlení. Zároveň se vypočítá člen $\frac{1}{N}$ uložený v konstantě `ref_y_`, který se používá při převodu do sRGB. Funkce `GetRGB` provádí převod ze vstupních spektrálních vzorků do sRGB barevného prostoru tak, jak bylo zmíněno v sekci 3.8.3.

5.6 Implementace pro GPU

Pro zjednodušení kontroly chyb byla implementace funkce `checkRTError`, která jako parametr bere `RTresult`. Není-li jeho hodnota `RT_SUCCESS`, vypíše se text chyby, ke které došlo. Volání jakékoliv funkce, která má návratovou hodnotu typu `RTresult`, je tedy obaleno touto kontrolou chyb.

Prvním krokem při GPU implementaci, pomineme-li načítání dat a inicializaci kamery, je vytvoření OptiX kontextu. U něj se nastavují následující parametry:

- Velikost zásobníku - experimentálně nastavena na 4096 bytů. Při nižších hodnotách docházelo k přetékání.
- Počet paprsků - v implementaci se používají 2 druhy paprsků. Pro výpočet osvětlení a kontrolu stínů.
- Počet vstupních bodů - používá se pouze jeden pro generování paprsků.
- Povolení výpisu - nastavení umožní použít funkci `rtPrintf` uvnitř CUDA kernelů. Tato funkce se používá pro výpis výjimek v patřičném programu.

Dále se vytváří kontextová proměnná `max_depth`, která definuje maximální hloubku zanoření rekurze. Také se specifikuje výstupní buffer, do kterého se ukládají data z trasování. Tento je trojrozměrný, kdy šířka a výška jsou dány požadovaným rozlišením výstupního obrazu, hloubka je pak nastavena na 81. Ukládají se tak veškeré finální spektrální vzorky. Formát prvku je `RT_FORMAT_FLOAT`.

Dále se vytvoří datové buffery, které drží informace o scéně a nahrají se do nich data. Těmito buffery jsou *vertex buffer*, *indices buffer*, *material buffer* a *light buffer*. Všechny jsou jednorozměrné. Zároveň jsou pro ně vytvořeny příslušné kontextové proměnné, na které jsou tyto buffery po naplnění daty navázány. Zde je přehled těchto bufferů, jejich formátování a využití:

- vertex buffer - `RT_FORMAT_USER` - jsou v něm uloženy všechny vertexy ve scéně, které jsou definovány strukturou `Vertex`. Velikost jednoho prvku je tedy velikost této struktury.
- indices buffer - `RT_FORMAT_UNSIGNED_INT4` - obsahuje čtveřice indexů (v_1, v_2, v_3, m) popisující jeden trojúhelník. Indexy v_i ukazují na vrcholy trojúhelníků ve vertex bufferu, m je index materiálu v material bufferu.
- material buffer - `RT_FORMAT_USER` - obsahuje struktury materiálů
- light buffer - `RT_FORMAT_USER` - obsahuje bodová světla, velikost prvku je tedy nastavena podle struktury `PointLight`.

Když jsou nahrána data, vytvářejí se OptiX materiály. Tyto materiály, jak bylo zmíněno v kapitole 4, představují funkce pro výpočty osvětlení pro jednotlivé druhy paprsků. Stejně jak na CPU, i na GPU je 5 různých materiálů pro různé BRDF. Jejich význam se od CPU implementace neliší. Každý tento materiál je implementován v samostatném .cu souboru, načítá se tedy z patřičného .PTX. Vytváření materiálu pak probíhá následovně. Nejprve se materiál inicializuje. Poté se z .PTX načtou closest a any hit programy. Ty se pomocí patřičných funkcí přiřadí materiálu. Closest hit je přiřazen paprskům osvětlení a any hit je přiřazen paprskům stínů. Všechny tyto materiály se zároveň ukládají do zásobníku, protože se dále přiřazují.

Poté se vytváří geometrická hierarchie. Nejprve je vytvořena samotná geometrie, které se z `geometry.cu.ptx` načtou a přiřadí bounding box a intersection programy. Následně je vytvořena geometry instance. Té se nastaví předchozí geometrie, určí počet materiálů a jednotlivé materiály nastaví. Tyto materiály se přiřazují z předem vytvořeného zásobníku. Poté se vytváří akcelerační struktury. V rámci volby algoritmu konstrukce jsem zvolil *Bvh*. Při nastavení *Sbvh* byly výsledky trasování neúplné, zřejmě proto, že dochází k úpravě index bufferu. OptiX však zřejmě nedokáže zapsat i dodatečná data (index materiálu pro trojúhelník), což vede k oné neúplnosti. Všechny tyto části se pak sloučí v geometry group. U ní se nastaví počet potomků, který je pouze jeden a přiřadí se. Dále se přiřazuje vytvořená akcelerační struktura.

Poté se nahrává program obsluhující vstupní bod pro trasování. Tento program generuje paprsky podle dostupných indexů. Obsahuje řadu proměnných:

- samples - určuje počet vzorků na pixel pro účely supersamplingu
- eye - pozice kamery v prostoru
- **u** - osa **x** souřadného systému kamery
- **v** - osa **y** souřadného systému kamery
- **w** - osa **z** souřadného systému kamery

Následně se program přiřadí kontextu k danému vstupnímu bodu. Zároveň se zde načítá exception program, který se také přiřadí vstupnímu bodu a miss program, který se přiřadí paprskům osvětlení.

Nyní se může spustit kontext. Když je trasování dokončeno, data se z GPU zkopírují do RAM paměti, přepočtou pomocí třídy `ObserverData` do sRGB a zobrazí.

5.7 Testy

Nyní se podíváme na naměřené výsledky při testování aplikace. V rámci předmětu Programování v prostředí HPC (PvPHPC) jsem měl možnost otestovat CPU implementaci na clusteru Anselm díky dočasnému studentskému přístupu. Tak jsem mohl otestovat, jak dobře se řešení škáluje napříč více výpočetními uzly. Prvním krokem bylo určit optimální poměr OpenMPI procesů a OpenMP vláken pro jeden uzel. Hardwarové specifikace uzlu

jsou následující: 2×Intel Sandy Bridge E5-2665 @ 2.4 GHz (celkem 16 jader), 64 GB @ 1600 Mhz DDR3 RAM.

Nastavení pro test bylo následující: rozlišení 320×180px, bez supersamplingu. Testovací scéna byla Cornell Box, viz obrázek 18. Naměřené časy jsou v sekundách. Pro každou kombinaci jsem provedl pět měření, které se nakonec zprůměrovaly

Kombinace	Měření 1	Měření 2	Měření 3	Měření 4	Měření 5	Průměr
1× MPI, 16× OMP	235.1 s	234.8 s	234.8 s	234.8 s	235.0 s	234.9 s
2× MPI, 8× OMP	168.9 s	167.7 s	167.8 s	168.1 s	168.1 s	168.1 s
4× MPI, 4× OMP	158.6 s	156.3 s	156.9 s	156.6 s	158.7 s	157.4 s
8× MPI, 2× OMP	172.3 s	172.2 s	172.9 s	172.2 s	173.1 s	172.6 s
16× MPI, 1× OMP	216.4 s	216.2 s	217.9 s	216.5 s	216.4 s	216.7 s

Tabulka 2: Časy výpočtu pro různé kombinace MPI/OMP v rámci jednoho uzlu

Jak je zřejmé, ideální kombinace jsou 4 OpenMPI procesy, kde každý běží ve 4 OpenMP vláknech, na jeden uzel. S tímto nastavením jsem následně testoval škálovatelnost na klastru přes jeden až čtyři uzly. Nastavení scény bylo stejné, časy jsou opět v sekundách.

Počet uzlů	Měření 1	Měření 2	Měření 3	Měření 4	Měření 5	Průměr
1	158.6 s	156.3 s	156.9 s	156.6 s	158.7 s	157.4 s
2	91.6 s	91.3 s	91.4 s	91.4 s	91.5 s	91.5 s
3	75.7 s	75.7 s	76.1 s	75.4 s	76.1 s	75.8 s
4	58.8 s	58.7 s	58.8 s	58.2 s	58.6 s	58.6 s

Tabulka 3: Tabulka škálovatelnosti CPU implementace napříč clusterem

Z výsledků je vidět, že škálovatelnost není lineární. To bude s největší pravděpodobností způsobeno režií na správu vláken/procesů a komunikaci mezi nimi.

Nyní se podívejme na testy GPU implementace. Ta byla otestována na dvou různých akcelerátorech a pěti různých scénách. První testovací sestavou byl můj domácí PC, jehož specifikace je následující

GPU: NV GeForce GTX 560Ti – 384 jader @ 900 MHz, 1 GB GDDR5, 1.26 TFLOPS, CUDA 2.1
CPU: AMD Phenom II X6 1075T – 6 jader @ 3512 MHz
RAM: 16 GB @ 1333 MHz, DDR3

Dále jsem provedl testy u pana Ing. Petra Gajdoše, Ph.D. na jeho PC

GPU: NV GeForce GTX Titan – 2688 jader @ 837 MHz, 6 GB GDDR5, 4.5 TFLOPS, CUDA 3.5
CPU: AMD FX-8150 – 8 jader @ 3610 MHz
RAM: 32 GB @ 1866 MHz, DDR3

Testovací scény vypadaly následovně:

Cornell box (obr. 18): jednoduchá scéna, uprostřed se nachází skleněná koule, v pravém rohu je opřené zrcadlo, v levém rohu stojí železná kostka. Tyto objekty jsou uvnitř místnosti, která je obarvena s využitím průběhů selektivní absorpce. Slouží pro ukázání obecné funkčnosti implementace.
Cornell box 2 (obr. 19): modifikace předchozí scény. V prostoru se nachází 2 dielektrické kvádry s různou tloušťkou, demonstrující funkčnost Beer-Lambertova zákona a jeho vliv na zbarvení dielektrik. Kvádr vlevo je přibližně 5× širší než ten vpravo. Jako materiál pro ně byl zvolen safír.
Gems (obr. 20): druhá modifikace Cornell boxu. V místnosti jsou tři drahokamy. Diamant, safír a smaragd. Opět slouží k demonstraci Beer-Lambertova zákona.
Dragon (obr. 21): scéna, kde na rovinné ploše stojí Stanfordský drak, jehož materiál je sklo. Model byl vybrán kvůli své vyšší komplexnosti.
Dragon – detail (obr. 22): Stejně jako předchozí scéna, kamera je však přiblížena na břicho draka, na kterém je vidět chromatická aberace.

Nastavení pro trasování bylo pro různé scény rozdílné, přesněji byly dvě sady:

Cornell box, Cornell box 2 a Gems: rozlišení 1600×900 px, 16× supersampling, maximální hloubka zanoření 10
Dragon a Dragon – detail: rozlišení 1600×900 px, 4× supersampling, maximální hloubka zanoření 10

Nutnost těchto rozdílů je dána hardwarovým omezením, přesněji velikostí paměti na grafické kartě. U scén s drakem totiž dochází k velké hloubce rekurze a karta nemá dostatek paměti, aby mohla provést šestnáctinásobné vzorkování na každý pixel.

V následující tabulce jsou konkrétní časy trasování.

Karta	Cornell Box	Cornell Box 2	Gems	Dragon	Dragon – detail
GTX 560Ti	46.6 s	40.3 s	114.5 s	466.6 s	1139.4 s
GTX Titan	18.9 s	18.1 s	46.3 s	183.6 s	458.7 s

Tabulka 4: Tabulka časů výpočtů GPU implementace

Podle HW specifikací jednotlivých akcelerátorů lze očekávat teoretické zrychlení přibližně 3.6× ve prospěch karty GTX Titan. Skutečné zrychlení je však v průměru cca 2.4×. Důvodem, že není dosaženo předpokládané teoretické zrychlení, může být nízká složitost úloh, takže výpočetní kapacita karty Titan nebyla plně využita.

6 Závěr

Cílem práce bylo seznámení se spektrálním trasováním. To zahrnuje znalost barevných prostorů a převodů mezi nimi, optických zákonů a chování světla v prostoru a optické vlastnosti materiálů. Druhou částí práce byla implementace spektrálního trasování. Byla implementována CPU i GPU verze algoritmu. Výsledky práce jsou:

1. Třída popisující standardního pozorovatele, která převádí spektrální vzorky do sRGB barevného prostoru.
2. Funkce pro načítání potřebných dat: 3D geometrie z formátu Wavefront .OBJ, spektrální data pro materiály ve vlastním jednoduchém formátu a načítání spektrálně definovaných světel. Tyto funkce data načtou do struktur vhodných pro výpočty.
3. Implementace spektrálního trasování pro CPU s využitím knihoven OpenMP a OpenMPI, umožňující distribuci výpočtů na libovolný počet výpočetních uzlů.
4. Implementace spektrálního trasování pro GPU založená na NVIDIA OptiX SDK.

Tyto výsledky tvoří solidní základ pro syntetizaci fotorealistických obrazů. V budoucnu bych práci zkusil dále vylepšit. Pro CPU i GPU implementaci existuje několik společných vylepšení. Prvním z nich by byla hybridní implementace s využitím polychromatických a monochromatických paprsků, díky níž by mělo dojít ke zrychlení výpočtů. Také bych chtěl přidat podporu pro více formátů 3D modelů. Pro lepší kvalitu výsledků by byla užitečná implementace techniky mapování normál, která učiní povrch hrbolatější. Samozřejmostí by byly další optimalizace programu, paměťové i výpočetní. V CPU implementaci by bylo možné například použit lepší konstrukční algoritmus pro akcelerační strukturu, využití vektorových instrukcí a celkové zlepšení programové struktury. Také by bylo vhodné implementovat GUI pro jednodušší zacházení s aplikací. GPU implementace by šla rozšířit o podporu akcelerace přes více grafických karet, což OptiX SDK umožňuje. Také by se daly optimalizovat datové struktury tak, aby bylo možné využít modernějších algoritmů konstrukce BVH, tj. SBVH a TRBVH.

Dále se chci pokusit o implementaci spektrálního path tracing, které ještě zvyšuje kvalitu výsledku díky simulací jevů, jako jsou například měkké stíny. Kvůli značně vyšším výpočetním nárokům této metody však bude potřeba nejprve provést výše zmíněné optimalizace.

Díky této práci jsem získal spoustu zajímavých znalostí z oblasti, která mne velmi zajímá a dle mého názoru má před sebou velkou budoucnost.

Bc. Jan Šurík

7 Reference

[1] Sojka, Eduard, *Počítačová grafika II*.

[2] cie.co.at, *CIE Divisions*, [ONLINE],
http://www.cie.co.at/index.php/Divisions

[3] cie.co.at, *CIE Free Documents for Download*, [ONLINE],
http://www.cie.co.at/index.php/LEFTMENUE/DOWNLOADS

[4] Smith, Thomas, a Guild, John, *The C.I.E. colorimetric standards and their use*, Transactions of the Optical Society 33, 1931. pp. 73 – 134
http://dx.doi.org/10.1088%2F1475-4878%2F33%2F3%2F301

[5] brucelindbloom.com, *RGB/XYZ Matrices*, [ONLINE],
http://www.brucelindbloom.com/index.html?Eqn_RGB_XYZ_Matrix.html

[6] Parker, Steven G. et al., *OptiX: A General Purpose Ray Tracing Engine*, NVIDIA,
http://graphics.cs.williams.edu/papers/OptiXSIGGRAPH10/Parker10OptiX.pdf

[7] Stich, Martin, Friedrich, Heiko, Dietrich, Andreas, *Spatial Splits in Bounding Volume Hierarchies*, NVIDIA Research,
http://www.nvidia.com/docs/IO/77714/sbvh.pdf

[8] Karras, Tero a Aila, Timo, *Fast Parallel Construction of High-Quality Bounding Volume Hierarchies*, NVIDIA,
https://research.nvidia.com/sites/default/files/publications/karras2013hpg_paper.pdf

[9] www.wikipedia.org, *Lidské oko*, [ONLINE],
http://cs.wikipedia.org/wiki/Lidsk%C3%A9_oko

[10] www.wikipedia.org, *CIE 1931 XYZ Color space*, [ONLINE],
http://en.wikipedia.org/wiki/CIE_1931_color_space

[11] www.wikipedia.org, *sRGB*, [ONLINE],
http://cs.wikipedia.org/wiki/SRGB

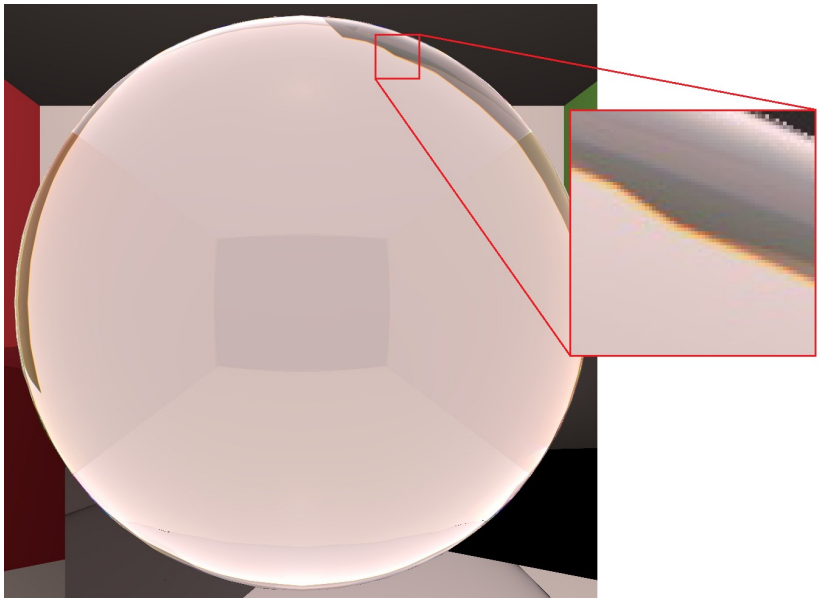
[12] www.baylee-online.net, *Refractive Indices*, [ONLINE],
http://www.baylee-online.net/Projects/Raytracing/Algorithms/Refractive-Indices

A Deklarace třídy BVH

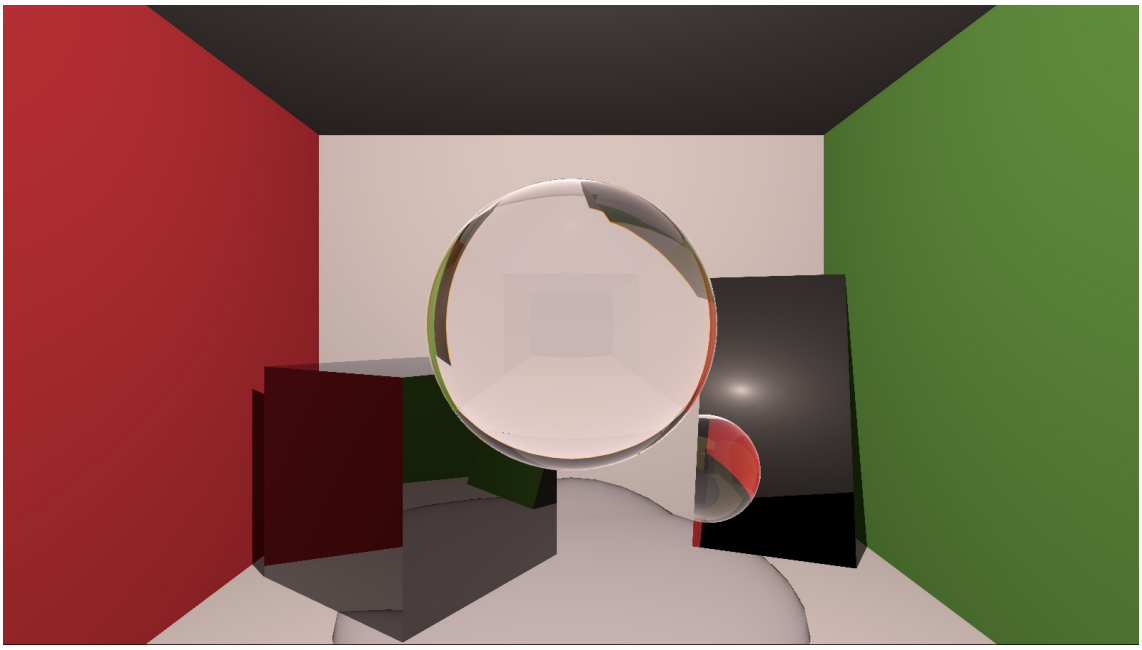
```
class BVH
{
public:
  BVH( void );
  BVH( vector<Vertex> & vertex_buffer, vector<triangle> & index_buffer, vector<Material> &
       material_buffer, vector<PointLight> & light_buffer, unsigned int max_depth, int
       max_leaf_items );
  ~BVH( void );
  float SpectralTrace( Ray * ray );

private:
  // Root of the hierarchy tree
  BVHNode * root_;

  // Data holders
  vector<Vertex> vertex_buffer_;      // Vertices storage
  vector<triangle> index_buffer_;     // Indices storage
  vector<Material> material_buffer_;  // Materials storage
  vector<PointLight> light_buffer_;   // Materials storage
  unsigned int max_depth_;

  // Construction constants
  int max_leaf_primitives_;
  float intersection_cost_;
  float traversal_cost_;

  // Construction
  BVHNode * BuildTree( int from, int to, int axis );
  void Sort( int from, int to, int axis );

  // SAH
  BVHNode * BuildSAH( int from, int to );
  float CalculateISA( AABB bounds );
  float CalculateSA( AABB bounds );

  // Traversal
  void Traverse( Ray * ray, BVHNode * node );
  bool RayBoxIntersection( Ray * ray, AABB * box );
  bool RayTriIntersection( Ray * ray, int tri_index_ );

  // Shading
  float Diffuse( Ray * ray, Vec3 hit_point, Vec3 hit_normal );
  float Phong( Ray * ray, Vec3 hit_point, Vec3 hit_normal );
  float Mirror( Ray * ray, Vec3 hit_point, Vec3 hit_normal );
  float Dielectric ( Ray * ray, Vec3 hit_point, Vec3 hit_normal );
  bool ShadowTest( Ray * ray, BVHNode * node, float light_norm );
};
```

Výpis 14: Třída obsluhující akcelerační strukturu

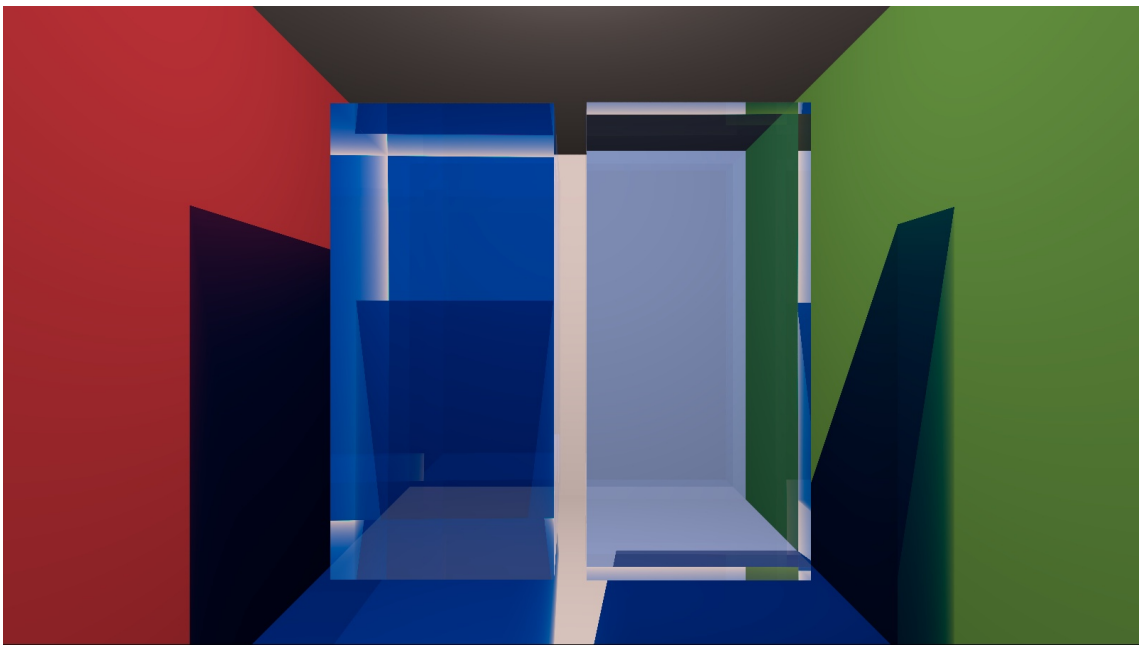
B Ukázky výsledků implementace



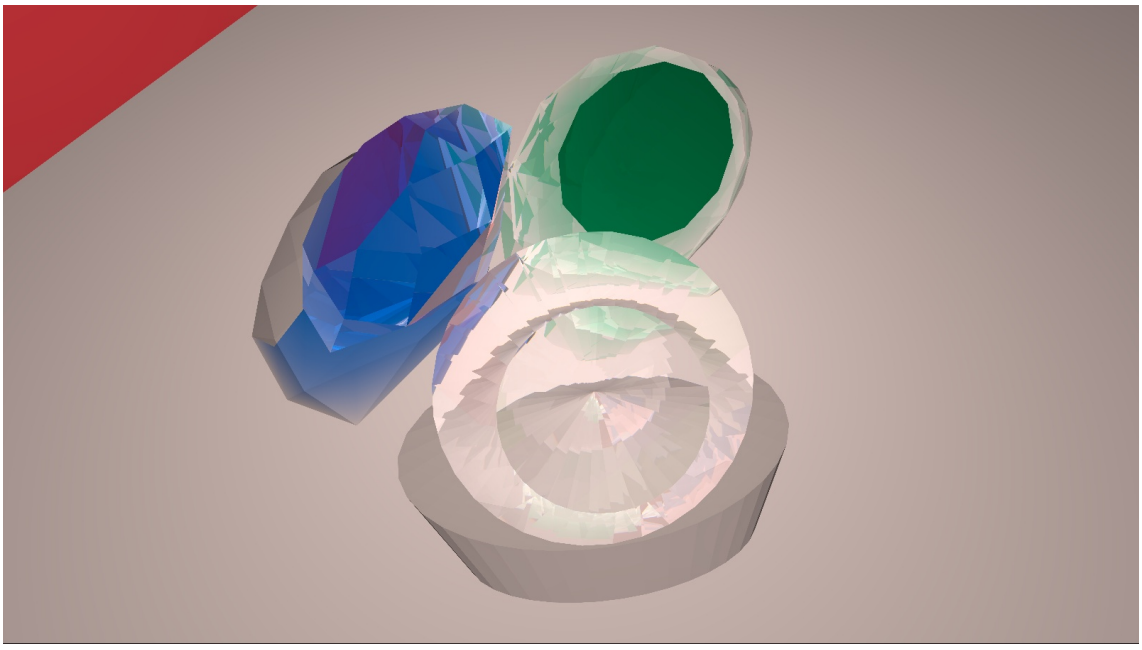
Obrázek 17: Detail chromatické aberace



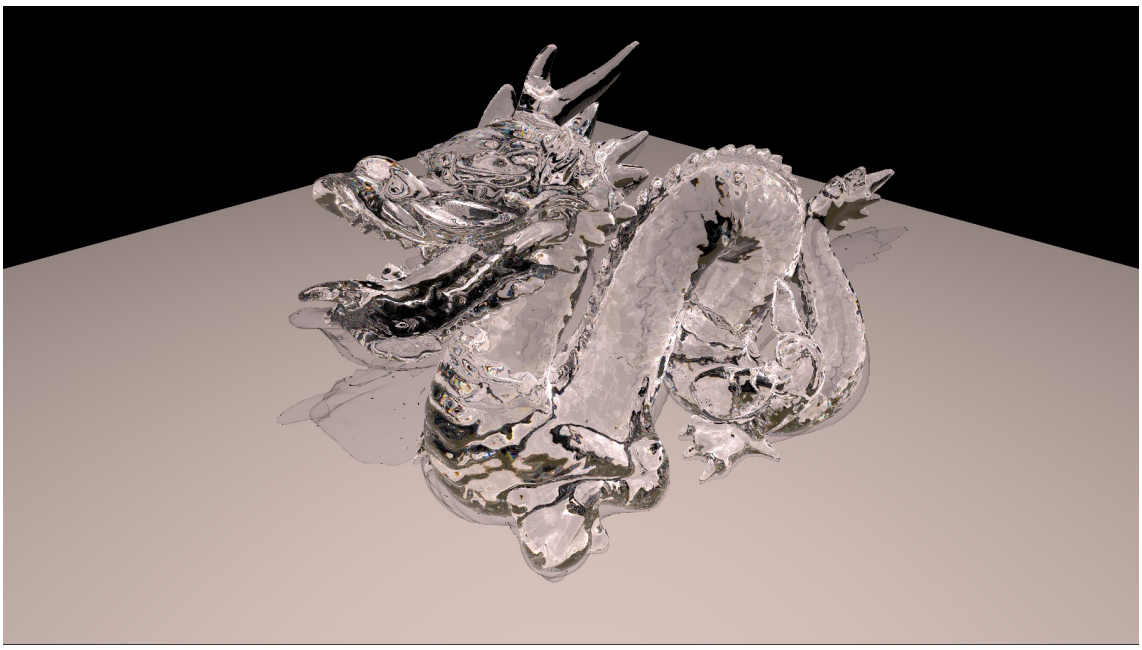
Obrázek 18: Scéna Cornell box. Uprostřed je skleněná koule, v níž se projevuje chromatická aberace. Po pravé straně je zrcadlový povrch, nalevo pak železná kostka.



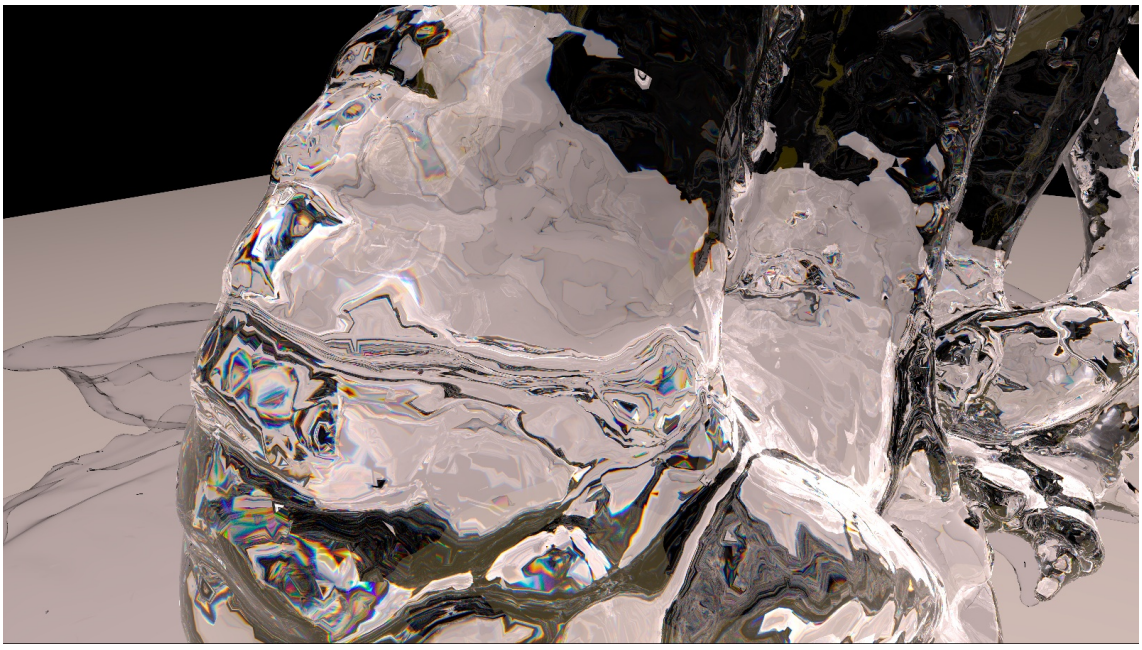
Obrázek 19: Scéna Cornell box 2. Ukázka vlivu Beer-Lambertova zákona. Oba objekty mají stejný materiál - safír. Objekt vlevo je přibližně 5× tlustší než objekt vpravo.



Obrázek 20: Scéna Gems. Ve scéně se nachází tři drahokamy. Jsou to diamant, safír a smaragd. Na safíru je vidět jeho zbarvení. U smaragdu již není tak patrné, což bude pravděpodobně kvůli chybě v datech materiálu.



Obrázek 21: Scéna Dragon. Ve scéně je model čínského draka ze Stanfordské univerzity. Modelu bylo jako materiál přiřazeno sklo.



Obrázek 22: Scéna Dragon. Detail na břicho stanfordského draka. Je zde jasně vidět chromatická aberace v opticky složitém prostředí.